

Using Group Role Assignment to  
Solve Dynamic Vehicle Routing Problem

by

Bo Lei

A thesis submitted in partial fulfillment  
of the requirements for the degree of  
Master of Science (MSc) in Computational Science

The Faculty of Graduate Studies  
Laurentian University  
Sudbury, Ontario, Canada

© Bo Lei, 201

**THESIS DEFENCE COMMITTEE/COMITÉ DE SOUTENANCE DE THÈSE**  
**Laurentian University/Université Laurentienne**  
Faculty of Graduate Studies/Faculté des études supérieures

Title of Thesis Titre de la thèse	Using Group Role Assignment to Solve Dynamic Vehicle Routing Problem	
Name of Candidate Nom du candidat	Lei, Bo	
Degree Diplôme	Master of Science	
Department/Program Département/Programme	Computational Sciences	Date of Defence Date de la soutenance September 04, 2018

**APPROVED/APPROUVÉ**

Thesis Examiners/Examineurs de thèse:

Dr. Youssou Gningue  
(Co-Supervisor/Co-directeur de thèse)

Dr. Haibin Zhu  
(Co-Supervisor/Co-directeur de thèse)

Dr. Matthias Takouda  
(Committee member/Membre du comité)

Dr. Dongning Liu  
(External Examiner/Examineur externe)

Approved for the Faculty of Graduate Studies  
Approuvé pour la Faculté des études supérieures  
Dr. David Lesbarrères  
Monsieur David Lesbarrères  
Dean, Faculty of Graduate Studies  
Doyen, Faculté des études supérieures

**ACCESSIBILITY CLAUSE AND PERMISSION TO USE**

I, **Bo Lei**, hereby grant to Laurentian University and/or its agents the non-exclusive license to archive and make accessible my thesis, dissertation, or project report in whole or in part in all forms of media, now or for the duration of my copyright ownership. I retain all other ownership rights to the copyright of the thesis, dissertation or project report. I also reserve the right to use in future works (such as articles or books) all or part of this thesis, dissertation, or project report. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that this copy is being made available in this form by the authority of the copyright owner solely for the purpose of private study and research and may not be copied or reproduced except as permitted by the copyright laws without written authority from the copyright owner.

## **Abstract**

The Dynamic Vehicle Routing Problem (DVRP) is a more complex problem than the traditional Vehicle Routing Problem (VRP) in the combinatorial optimization of operations research. With more degrees of freedom, DVRP introduces new challenges while judging the merit of a given route plan.

This thesis utilized the time slice strategy to solve dynamic and deterministic routing problems. Based on Group Role Assignment (GRA) and two different routing methods (Modified Insertion heuristic routing and Modified Composite Pairing Or-opt routing), a new ridesharing system has been designed to provide services in the real world.

Simulation results are presented in this thesis. A qualitative comparison has been made to outline the advantages and performance of our solution framework. From the numerical results, the proposed method has a great potential to put into operation in the real world and provides a new transit option for the public.

### **Keywords:**

Dynamic Vehicle Routing, Ridesharing, Group Role Assignment, Modified Insertion Heuristic Routing, Modified Pairing Or-opt Routing.

## **Acknowledgments**

To Dr. Zhu and Dr. Youssou, no matter when I feel confused, they always guide me and give me the direction. Their rigorous academic attitude and profound knowledge deserve me to learn from all my life. Without their kind assistance, this thesis could not be possible.

I also want to thank Dr. Matthias Takouda of accepting to be a member of the committee.

I would like to express my deep appreciation to my family members, their love and support are the most precious things for me.

I am really grateful to Yifei Ren, my brother, no matter when I need help, he is always ready to give his hands to me. The friendship with him has alleviated much home sick because of studying abroad.

A special thank you to John Rinaldo, my ‘landlord’, I could not remember how many nights I have lived in his house for free. I can clearly remember he helped me to tow my car out of snow in Canada’s extreme winter.

Also, I want to say thank you to Dr. Koczkodaj and Hayat Shamakhai for their kind help.

# Table of Contents

<b>Abstract.....</b>	<b>III</b>
<b>Acknowledgments .....</b>	<b>IV</b>
<b>Table of Contents .....</b>	<b>V</b>
<b>List of Tables .....</b>	<b>VII</b>
<b>List of Figures.....</b>	<b>VIII</b>
<b>List of Appendices.....</b>	<b>IX</b>
<b>Abbreviations .....</b>	<b>X</b>
<b>1 Introduction .....</b>	<b>1</b>
1.1 Background.....	2
1.2 Solution Strategy Analysis.....	3
<b>2 Group Role Assignment Problem.....</b>	<b>8</b>
2.1 Introduction.....	9
2.2 Related Development of the GAP.....	10
2.3 Formulation of the GAP.....	14
2.4 Convert the GRAP to the GAP .....	15
<b>3 Vehicle Routing Problem and Dynamic Vehicle Routing Problem.....</b>	<b>19</b>
3.1 Introduction of the VRP.....	20
3.2 Literature Review of the VRP.....	21
3.3 Introduction of the DVRP.....	27
3.4 Literature Review of the DVRP.....	30
<b>4 Dial-a-Ride Problem .....</b>	<b>34</b>
4.1 Formulation of the DARP.....	35
4.2 Main Features of the DARP.....	36

4.3	Review of the Single Vehicle DARP .....	37
4.4	Review of the Multi-Vehicle DARP .....	39
<b>5</b>	<b>Algorithms .....</b>	<b>44</b>
5.1	The Proposed Algorithms .....	45
5.1.1	Phase I – Assign passengers to buses .....	45
5.1.2	Phase II – Design the routes for buses .....	47
5.1.3	Execute the routes and update the information .....	53
5.2	Platform of simulation .....	54
<b>6</b>	<b>Simulation Results and Analysis .....</b>	<b>56</b>
6.1	Instance generation .....	57
6.2	Simulation results of different coefficients .....	57
6.3	Simulation results of different routing algorithms .....	58
6.4	Equal pressure test .....	59
6.5	Differential pressure test .....	60
6.6	Qualitative Comparison .....	64
<b>7</b>	<b>Conclusion .....</b>	<b>68</b>
	<b>References .....</b>	<b>72</b>
	<b>Appendix I .....</b>	<b>85</b>
	<b>Appendix II .....</b>	<b>95</b>
	<b>Appendix III .....</b>	<b>103</b>
	<b>Appendix IV .....</b>	<b>119</b>

## List of Tables

Table 6.1: The comparison of using different coefficient datasets.....	58
Table 6.2: The comparison of using different routing algorithms.....	59
Table 6.3: The numerical results of equal-pressure test using routing algorithm III.....	60
Table 6.4: The numerical results of equal-pressure test using routing algorithm IV.....	60
Table 6.5: The numerical results of differential-pressure test using routing algorithm III....	61
Table 6.6: The numerical results of differential-pressure test using routing algorithm IV...	61
Table 6.7: The numerical results of 20 datasets with differential-pressure test using routing algorithm III.....	62
Table 6.8: The numerical results of 20 datasets with differential-pressure test using routing algorithm IV.....	63
Table 6.9: The qualitative comparison results of different algorithms.....	66

## **List of Figures**

Figure 3.1: Example of dynamic vehicle routing.....	28
Figure 5.1: The overall program flow chart.....	54



## **List of Appendices**

Appendix I: Dataset of total 400 requests generated randomly.....	85
Appendix II: Simulation results of differential pressure test.....	95
Appendix III: An example to demonstrate the key mechanism.....	103
Appendix IV: The main code used in the simulation.....	119

## **Abbreviations**

GRAP	Group Role Assignment Problem
RBC	Role-Based Collaboration
GAP	Generalized Assignment Problem
AP	Assignment Problem
VRP	Vehicle Routing Problem
TSP	Travelling Salesman Problem
VRPPD	Vehicle Routing Problem with Pickup and Delivery
DVRP	Dynamic Vehicle Routing Problem
DARP	Dial-a-Ride Problem

# Chapter 1

## 1 Introduction

This chapter will introduce some background information and solution strategy analysis of this thesis. This will include:

- The background of our research problem,
- The model of our solution framework,
- The analysis of solution strategy of this area, and
- The potential improvements made.

## 1.1 Background

A public transportation system plays a significant role in the urban life, it is essential for everyone living in the cities. Consequently, the efficiency of a public transit system is one of the most important factors that citizens concern about. However, with the different scales of cities, transit systems differ greatly. Take Toronto as an example, it is a huge metropolitan in Ontario. TTC (Toronto Transit Commission) needs to service millions of trips every day. Compared with Toronto, North Bay, a relatively small city in the northern Ontario, has an extremely compacted transit system. There are only 22 buses in total, and only 13 of them put into operation. It is reasonable because the number of people serviced in North Bay is much smaller than in Toronto. Even though the transit system works well in North Bay, improvement still can be achieved. In the off-peak period, there are not many requests need to be served in the city. But all 13 buses still have to run on the fixed routes, sometimes only 1 or 2 passengers in a bus. It is really a waste of public transportation power and a lack of optimization strategy.

With the rapid development of information technology, on-demand transportation system becomes more and more popular. Utilizing limited vehicles, it provides passengers a more efficient commuting method. Nowadays, there are several well-known ride-sharing, app-based transportation systems and companies, such as Uber and DiDi. In the daily life, an optimized approach is put into practice to manipulate the complex supply and demand assignment tasks and obtain many benefits for both the travelers and transport supporters. From the perspective of travelers, with the help of application, their waiting time can be reduced and they can control their schedules more leisurely, because their exact pickup time can be estimated accurately in the application. From the standpoint of transport supporters, based on the demands of passengers, they

can exploit all the transportation power to provide better service to all the users. From the environment protection, it can increase the fuel efficiency and decrease the overall energy use, reduce the pollution at the same time.

The ride-sharing problem can be mathematically modelled by the famous optimization problem which is Capacitated Vehicle Routing Problem with Simultaneously Pickup and Delivery (CVRPSPD). For the public transit system, it is more complicated because of the scale and the time uncertainty of demands.

This thesis proposes a new heuristic solution framework trying to turn the traditional transit system into a ride-sharing on-demand transportation system so as to reduce the total time-consuming of all the passengers and mileage of vehicles.

## 1.2 Solution Strategy Analysis

The Vehicle Routing Problem(VRP) formulation was first introduced by Dantzig and Ramser [1], and is generally defined on a graph  $G = (V, \mathcal{E}, C)$ , where  $V = \{v_0, \dots, v_n\}$  is the set of vertices;  $\mathcal{E} = \{(v_i, v_j) | (v_i, v_j) \in V^2, i \neq j\}$  is the arc set;  $C = \{(C_{ij}) | (v_i, v_j) \in \mathcal{E}\}$  is the cost matrix, representing distances, time cost or travel cost.

Compared with the classical definition of the VRP, real world applications often include an important factor, the evolution of information, which relates to the fact that in some problems the information available to the planner may change during the execution of the routes, such as the arrival of new passenger requests. Besides, based on the problem and the available technology, vehicle routes can either be designed statically or dynamically.

Indeed, in the public transit system, demands arrive randomly in time, have a random size, and thus routing is a continuous process of collecting demands, forming tours, and dispatching vehicles if turning it to a ride-sharing on-demand transportation system. Consequently, this problem is no longer a static and deterministic one, it becomes a dynamic and deterministic problem. Part of the input (new requests) is unknown and revealed dynamically during the design or execution of the routes. The requests are deterministic once they are revealed. For this sort of problems, vehicle routes are redefined in an ongoing fashion, requiring technological support for real-time communication between the vehicles and the decision maker (e.g., mobile phones and global positioning system) [2].

In the dynamic and deterministic routing problems, crucible information is revealed over time, meaning that the complete instance is only known at the end of the planning horizon. As a consequence, an optimal solution can only be found a-posteriori, and exact methods only provide an optimal solution for the current state, lacking of any guarantee that the solution will be the optimal once new data becomes available. And there is a main drawback of exact methods that it cannot be applied to large instances. The related review is in Chapter 3.

Because optimal solutions can only be found a-posteriori, most dynamic approaches rely on heuristic approaches able to compute quickly a solution to the current state of the problem. The review is also in Chapter 3. As the public transit system always needs to provide service to the passengers for a period of time, redesigning the routes when every new request received have several difficulties with such an approach: computing resources might not be available, redesign the routes might be very time-consuming, it may make drives confusion etc. Like the traditional transit system, which partitions all the buses into different schedules, this period of service time

can be partitioned into relatively small segments (like every 15 minutes as a segment). A request arriving during a time segment is not handled until the end of the time bucket, thus the problem solved during a time slice only considers the requests known at its beginning. Then the system in each segment can be regarded as a CVRPSPD if we do not consider the new requests gathered within this segment. Hence, the optimization is run statically and independently during each time slice.

There are a number of exact and approximate algorithms have been developed for the VRP, the CVRPSPD is a variant of it. The review is in Chapter 3. Exact algorithms can only solve relatively small problems, but a number of approximate algorithms have proved very satisfactory. Heuristic algorithms can compute the solutions quickly and have potentials to be applied in solving the large problems.

There are different heuristic approaches, considering the scale and the limited transportation resource of the public transit system, two-phase methods need special interest. These heuristics are of two types: cluster first-route second, or route first-cluster second. In the first category, customers are clustered into groups and assigned to vehicles (phase I) and then efficient routes are designed for each cluster (phase II). In the second category, one constructs a traveling salesman tour through all the customers (phase I) and then partitions the tour into segments (phase II). One vehicle is assigned to each segment and visits the customers according to their appearance on the traveling salesman tour. Bertsimas and Simchi-levi [3] has proven the empirically well-studied route first-cluster second methods can never be asymptotically optimal for the capacitated VRP with unsplit demands except in some trivial cases. On the other hand, cluster first-route second has a different scenario. Location-based heuristic (LBH) can be

asymptotically optimal, which was proven by Bramel and Simchi-Levi [4]. The generalized assignment heuristic of Fisher and Jaikumar [5] can be viewed as a special case of the LBH in which the seed customers are first selected by a dispatcher. In the second step, customers are assigned to the seeds in an efficient way by solving a generalized assignment problem (GAP). The advantage of the LBH is that the selection of the seeds and the assignment of customers to seeds are done simultaneously, and not sequentially as in the generalized assignment heuristic. Therefore, when the generalized assignment heuristic is carefully implemented, it is asymptotically optimal as well.

Even though the generalized assignment heuristic has achieved impressive performance and always can get solutions, improvements still can be made. In the generalized assignment heuristic, the focus is on the minimize the total length of routes, without caring about the capacity of vehicles and the time consuming of customers, which are two significant factors in the public transit system.

Group Role Assignment Problem (GRAP) was first put forward by Zhu et al. [6] in 2012, which can be able to take these important factors into account. With the help of GRAP, it has the advantage of LBH coming from the generalized assignment heuristic, and also can always get solutions for assigning passengers to different vehicles. Following the definitions of GRAP, the agents are the passengers, the roles are the vehicles. Since the time slice will be used in our solution framework, this location-based heuristic will be dynamically changed with the running of vehicles in different time slices. Besides, GRAP can take the time cost of every passenger into account in the whole process, consequently, the user experience can be improved at the same time.



The remainder of this thesis is organized as follows, Chapter 2 reviews the GRAP. Chapter 3 is the review of VRP and dynamic VRP (DVRP). The review of Dial-a-ride Problem (DARP) is in Chapter 4. Chapter 5 is the algorithms we proposed and used. Simulation results are in Chapter 6. At last, conclusions and future work are provided in Chapter 7.

## Chapter 2

### 2 Group Role Assignment Problem

This chapter is the review of the Group Role Assignment Problem (GRAP), since it is related to the Generalized Assignment Problem (GAP), GAP is also included in this chapter. This chapter will include:

- The introduction of GRAP,
- The relationship between GRAP and GAP,
- The formulation of GAP,
- The related works of GAP,
- The formulation of GRAP,
- How to convert GRAP to GAP, and
- The solution of GRAP.

## 2.1 Introduction

In the practical world, everyone is not likely to be isolated from others. Everyone plays different roles in the daily life, such as son, parent, spouse, colleague, friend etc. Also, everyone has to interact with others, no matter at work, in the family or in a public place. Consequently, collaboration is necessary to meet everyone's needs and get the best performance of the whole system. Role-Based Collaboration (RBC) is an approach to facilitate an organizational structure, collaborate and coordinate activities of everyone with or within systems [7]. Take the transit system as an example, passengers have different demands, but the transportation power is limited. How to exploit the transportation power to give better service to every passenger and try best to reduce the budget is the ultimate purpose for the transit system. From this perspective, RBC is a useful and functional methodology.

In RBC, role assignment is a critical task that affects the efficiency of collaboration and the level of satisfaction of all the members involved in. Group Role Assignment Problem (GRAP) intimates a group by assigning roles to agents to achieve its best performance [6]. For example, in a company, if a manager wants to pick up 5 people from 10 in the team to be responsible for the marketing so that the performance of selected lineup can be maximized, how to do it is a typical GRAP.

GRAP can be converted to a Generalized Assignment Problem (GAP). As Kuhn-Munkres (K-M) algorithm has been designed to solve GAP with the complexity of  $O(m^3)$ , GRAP can also use this algorithm.

For GAP, the objective is to find an assignment in which all agents do not exceed their budget and the total profit of the assignment is maximized. The complexity of GAP is NP-Hard. Different approximation algorithms have been put forward to give approximate solutions for it.

In the remaining part of this chapter, the related development of GAP and how to convert a GRAP to a GAP will be briefly introduced.

## 2.2 Related Development of the GAP

Many approximation algorithms have been proposed to obtain a better solution for GAP. These algorithms can be divided into several categories:

- Branch and bound
- Branch and price
- Heuristic methods

For branch and bound, it approaches exhaustive enumeration with the efficient estimation of lower and upper bounds of every branch.

Ross and Soland [8] firstly developed a branch and bound algorithm to solve GAP by solving a series of binary knapsack problems to determine the bounds. Lagarangian Relaxation was used to get the lower bounds. Martello and Toth [9] used a maximization version of GAP, with deletion of the assignment constraints the relaxation was decomposed into a series of knapsack problems. The upper bound was obtained by the relaxation, the lower bound was calculated with the penalty for the violated assignment.

Fishes et al. [10] presented a branch and bound algorithm in which bounds are achieved from a Lagrangian relaxation with the multipliers set by a heuristic adjustment method. A comparison of their algorithm and those of Ross and Soland, Martello and Toth was made.

Guignard and Rosenwein [11] improved the approach of Fishes et al., at the root node they used subgradient optimization to solve the relaxation, the addition of the surrogate constraint strengthened the relaxation.

Nauss [12] utilized linear programming cuts, feasible-solution generators, Lagrangian relaxation and subgradient optimization into the branch and bound algorithm. With the help of two heuristics, good feasible solutions can be generated early in the process, which reduced the computation time largely. Based on the method of Nauss [12], Laguna et al. [13] used the Tabu Search heuristic to generate the initial feasible solution.

The branch and price method is a hybrid of branch and bound and the column generation methods.

Savelsbergh [14] firstly used this method to solve GAP. Caselli and Righini [15] utilized this algorithm to solve the multilevel GAP, based on a decomposition into a master problem with set-partitioning constraints and a pricing subproblem.

Based on the enumeration strategies, some problems still can not be solved in reasonable computation time. As a result, many heuristic approaches were designed to find high quality solutions.

As mentioned above, Laguna et al. [13] used the Tabu Search heuristic to generate the initial feasible solution by relaxing the capacity constraints and making assignments according to

minimum cost for a minimization problem. Diaz et al. [149] and Higgins [17] proposed two Tabu Search methods. The difference between them is that the former adopted a dynamic oscillation approach to adjust the penalty weight with the search going on, the latter used a simple dynamic Tabu Search strategy and focused on very large problems. Yagiura et al. [18] introduced the construction of three ejection chains in a local search phase, in the search process, penalty weights were adjusted dynamically.

Compared with Tabu Search, Genetic Algorithms(GA) used a random approach to simulate the process of evolution, which also were deployed to solve GAP.

Chu and Beasley [19] firstly applied GA to solve GAP. The initial solution was generated randomly and then the fitness would be assessed by computing two values, fitness and unfitness. Wilson [20] proposed an alternative GA method, which emphasized to get solutions with potentially optimal objective function values. After that, improve the feasibility of these solutions in terms of objective function value. The advantage is his method focused on improving feasibility and optimality simultaneously.

Raidl and Feltl [21] improved GA for solving GAP with the calculation of two variants in order to generate a high portion of feasible solutions in the initial population of solutions.

Simulated Annealing(SA) algorithms were also used to solve GAP as it simulates the annealing process of a solid and is a strategy can be used to guide a local search to find high quality local optimum. It was first used to solve GAP by Osman [22]. The algorithm presented combined the SA approach with Tabu Search to improve the solution gradually.

Path relinking method focuses on generating new solutions by combining attributes from two different solutions. Yagiura et al. [23] applied it to improve their ejection chain approach.

Amini and Racer [24] proposed a variable depth search procedure for GAP with a two-phase effort to improve solutions. In the first phase, an initial solution was generated randomly and the lower bound of objective function was obtained by solving the linear relaxation of GAP. In the second phase, improved solutions were achieved by generating the sequences of feasible task re-assignments which would cause the reduction of the objective function value. Then a hybrid heuristic method was reported by them by using the Martello and Toth [25] heuristic to generate high quality solutions and then refine them using variable depth search procedure. After that, a branching variable depth search approach was introduced by Yagiura et al. [26].

Trick [27] implemented a variable fixing approach to reduce the problem and then used the solution to the linear programming relaxation of GAP so as to set ‘futile’ variables to 0. A proof is also provided.

Lourenco and Serra [28] presented adaptive search heuristics for GAP. The generation of initial solutions was based on the MAX-MIN ant system heuristic and the greedy randomized adaptive search process.

Baykasoglu et al. [29] implemented the artificial bee colony algorithm to solve GAP.

## 2.3 Formulation of the GAP

The Generalized Assignment Problem is to either minimize or maximize the total profit while assign  $n$  jobs ( $i = 1, \dots, n$ ) to  $m$  agents ( $j = 1, \dots, m$ ) without exceeding their budget. One job only can be assigned to one agent.

GAP can be formulated as an integer program:

$$\left\{ \begin{array}{l} \max \sum_{i=1}^m \sum_{j=1}^n p_{ij} x_{ij} \\ \sum_{j=1}^n w_{ij} x_{ij} \leq W_i \quad i = 1, \dots, m; \\ \sum_{i=1}^m x_{ij} = 1 \quad j = 1, \dots, n; \\ x_{ij} \in \{0, 1\} \quad i = 1, \dots, m, \quad j = 1, \dots, n; \end{array} \right. \quad \begin{array}{l} (2.3.1) \\ (2.3.2) \\ (2.3.3) \\ (2.3.4) \end{array}$$

The decision valuable of the GAP is a binary variable  $x_{ij}$ , which is defined in (2.3.4). If  $x_{ij} = 1$ , it represents the job is assigned to the agent. Otherwise, if  $x_{ij} = 0$ , it represents the job is not assigned to the agent.

The parameters can be defined as follows.

The profit of assigning job  $j$  to agent  $i$  is represented by  $p_{ij}$ .

The weight of assigning job  $j$  to agent  $i$  is represented by  $w_{ij}$ .

The budget allocated for agent  $i$  is denoted by  $W_i$ .



The objective function is to maximize the total profit of all the assignments.

The first constraint (2.3.2) is the limit of budget.

The second constraint (2.3.3) ensures every agent is assigned exactly one job.

The third constraint (2.3.4) outlines the decision valuable and specify the ranges of both variables  $i$  and  $j$ .

GAP is a generalization of the Assignment Problem(AP). In the specific case in which the budgets of all the agents and the costs of all the tasks are equal to 1, GAP is reduced to AP. AP has been solved by Hungarian Method (also known as K-M Algorithm) in polynomial time [30] [31].

The 0-1 Multiple Knapsack problem is also a special case of GAP when item  $j$  is assigned to knapsack  $i$  with weight  $w_i$ , the budget is  $W_i$  and the profit is  $p_j$ .

## 2.4 Convert the GRAP to the GAP

GRAP was first presented by Zhu et al. [6] in 2012. In GRAP, the role can be assigned to more than one agent and the agent can receive only one role.

GRAP can be formulated as an integer program:

$$\left\{ \begin{array}{l} \max \sum_{i=1}^m \sum_{j=1}^n p_{ij} x_{ij} \quad (2.4.1) \\ \sum_{i=1}^m x_{ij} = L[j] \quad j = 1, \dots, n, \\ L[j] \in N, N \text{ is a set of natural numbers;} \quad (2.4.2) \\ \sum_{i=1}^m x_{ij} = 1 \quad j = 1, \dots, n; \quad (2.4.3) \\ x_{ij} \in \{0, 1\} \quad i = 1, \dots, m, \quad j = 1, \dots, n; \quad (2.4.4) \end{array} \right.$$

Compared with GAP, the first difference is on the objective function, the objective function of GRAP is usually formulated as a maximization problem. The second one is that the weight  $w_{ij}$  becomes uniform for all the agents and  $w_{ij} = 1$ . The unit profit  $p_{ij}$  is dependent of  $i$  and  $j$ .

The parameter  $i$  is allocated to the agent  $i$ .

The parameter  $j$  is allocated to the group role  $j$ .

The parameters  $L[j]$  ( $i = 1, \dots, m$ ) represent the minimum numbers of agents required for role  $j$ . They are integers.

GRAP is a well-known difficult problem, because it needs advanced methodologies for information classification, data mining, pattern search and matching [6].

However, Zhu et al. [6] proposed a solution for GRAP by using K-M algorithm. In the paper [6], how to convert GRAP to GAP was also demonstrated. They firstly introduced a role range vector to regulate the number of agents which can be assigned to different roles.

The role range vector is denoted as  $L[j] \in N$ , where  $N$  is a set of natural numbers.

The main step to convert GRAP to GAP is to adjust the number of agents and roles.

If  $m = \sum_{n=0}^{n-1} L[j]$ , the GRAP becomes a GAP. It means that GRAP can be converted to a GAP when the total value of role range vector is equal to the number of agents.

However, the input of K-M algorithm is a square matrix, which means the number of agents should be equal to that of roles. In the practical application, the assignments are not always following this condition. Consequently, transfer the input matrix into a square matrix is a prerequisite to using K-M algorithm.

In the paper [6], a transfer function was designed to fulfill this step. To form the square matrix used in the K-M algorithm, duplicate rows or columns should be added based on the role range vector  $L$ . The main idea is when the number of agents greater than roles, duplicate the columns to transfer to a square matrix, when the number of roles larger than that of agents, duplicate the rows to obtain a square matrix. After calling the K-M algorithm, the final assignment is produced based on the result.

Besides, Zhu et al. [6] also introduced the Rated GRAP (RGRAP) and the Weighted GRAP (WGRAP). The former assumes the roles are equally important, the value represents how well a given agent plays a given role. The latter can be viewed as an extension of the former, which assumes the roles in a group have different importance, i.e., the roles have different weights.

After that, numerical results of different sizes of assignments have also presented in this paper [6]. From the presented results, the group size varied from 10 to 100, with the efficiency of

K-M algorithm, it can solve the assignment within 40 milliseconds on the provided simulation platform. The computational complexity of the K-M algorithm is  $O(m^3)$ .

## **Chapter 3**

### **3    Vehicle Routing Problem and Dynamic Vehicle Routing Problem**

The Vehicle Routing Problem (VRP) and the Dynamic Vehicle Routing Problem (DVRP) are the combinatorial optimization problems in operations research. This chapter will provide

- The introduction of the VRP,
- The literature review of the VRP,
- The introduction of the DVRP, and
- The literature review of the DVRP.

### 3.1 Introduction of the VRP

The Travelling Salesman Problem (TSP) consists of finding the shortest path tour  $T$  between  $n$  cities. In term of graphs, it means finding the Hamiltonian cycle associated with the shortest total distance. The TSP, known in the theory of computational complexity as an NP-hard problem in combinatorial optimizations, plays a very important role in operations research and theoretical computer science.

The Vehicle Routing Problem (VRP) is a generalization of the well-known Travelling Salesman Problem. VRP is a combinatorial optimization and integer programming problem to find the optimal set of routes for a fleet of vehicles to traverse in order to deliver a set of customers.

The VRP is defined on a graph  $G = (V, \mathcal{E}, C)$ , where  $V = \{v_0, \dots, v_n\}$  is the set of vertices;  $\mathcal{E} = \{(v_i, v_j) | (v_i, v_j) \in V^2, i \neq j\}$  is the arc set;  $C = \{(C_{ij}) | (v_i, v_j) \in \mathcal{E}\}$  is the cost matrix, representing distances, time cost or travel cost.

The VRP consists of establishing minimum cost vehicle routes in such a way [32] that

- (i) each city in  $V$  is visited exactly once and by exactly one vehicle;
- (ii) all vehicles start and end at the depot; and
- (iii) some side constraints are satisfied.

The most common side constraints include:

- (i) capacity restrictions: a non-negative weight (or demand)  $d_i$  is attached to each city  $i > 1$  and the sum of weights of any vehicle route may not exceed the vehicle capacity. Capacity-constrained VRPs will be referred to as CVRPs;

- (ii) the number of cities on any route is bounded above by  $q$  (this is a special case of (i) with  $d_i = 1$  for all  $i > 1$  and  $D = q$ );
- (iii) total time restrictions: the length of any route may not exceed a prescribed bound  $L$ ; this length is made up of intercity travel times  $c_{ij}$  and of stopping times  $\delta_i$  at each city  $i$  on the route. Time- or distance-constrained VRPs will be referred to as TVRPs;
- (iv) time windows: city  $i$  must be visited within the time interval  $[a_i, b_i]$  and waiting is allowed at city  $i$ ;
- (v) precedence relations between pairs of cities: city  $i$  may have to be visited before city  $j$ .

This list is by no means exhaustive. In the next subsection 3.2, we will mainly concentrate on CVRPs and on TVRPs.

## 3.2 Literature Review of the VRP

In the last half century, VRP has attracted many operations researchers. This interest is mainly due to the practical importance of the problem, but also to its intrinsic difficulty. A number of exact and approximate algorithms have been proposed to solve the VRP.

For the exact algorithms, following the Laporte and Nobert [33] survey, it can be classified into three broad categories:

- (i) direct tree search methods;
- (ii) dynamic programming (DP); and
- (iii) integer linear programming (ILP).

Category (iii) is very broad and attracts most of the research effort these years. It can be divided into three sections:

(iiia) set partitioning formulations;

(iiib) vehicle flow formulations; and

(iiic) commodity flow formulations.

A direct tree search method consists of sequentially building vehicle routes by means of a branch and bound tree. It was first presented by Christofides and Eilon [34] and applies to CVRPs and TVRPs. This algorithm was tested on two VRPs with capacity restrictions only: one is 6-city problem and the other is Dantzig and Ramser 13-city example [1] were solved in 1.5 and 5 minutes respectively on an IBM 7090.

Christofides [35] described a depth first branch and bound algorithm based on a different philosophy. This algorithm can be applied to the problems include multiple constraints: capacity or distance restrictions, time windows, precedence conditions, stopping times etc. This method performs better on tight problems as less branches need to be explored. The largest problem solved with this algorithm contains 31 cities.

Christofides et al. [36] presented tree search algorithms for the exact solution of the VRP incorporating lower bounds computed from shortest spanning  $k$ -degree centre tree and  $q$ -routes. Their algorithm was applied to 25 customers and got the exact solution. Kolen et al. [37] used the concept of  $q$ -routes to derive an exact branch and bound algorithm for the CVRP with time windows. Nine test problems involving from 6 to 15 cities were solved to optimality.



For the DP, with the help of state-space relaxation, Christofides et al. [38] presented three formulations for the CVRP, he reported that CVRPs involving up to 50 nodes could be solved systematically by this approach.

Psaraftis [39] considered the static and the dynamic cases with capacity and maximum position shift constraints. He proposed an exact algorithm, an extension of the classical Held and Karp [40] DP algorithm for the TSP and succeeded in solving 9 customers problem in the static case. He later described a similar algorithm [41] using time windows to replace the maximum position shift constraint.

Desrosiers et al. [42] used a decomposition approach which provides a suboptimal solution and a problem of 880 requests was solved.

For the ILP, the first section is set partitioning formulations. Balinski and Quandt [43] were the first to propose a set partitioning formulation for VRPs. Their formulation has two main difficulties, the first one is the large number of variables and the other is the difficulty of computing the capacity. Rao and Zionts [44], Foster and Ryan [45], Orloff [46], Desrosiers et al. [47], Agarwal et al. [48] used a column generation algorithm to overcome these two difficulties. This method obtained the solution of VRPs with time windows containing up to 100 vertices.

Fisher and Jaikumar [5] [49] had developed a three-index vehicle flow formulation for VRPs with capacity restrictions, time windows and no stopping times  $\delta_i$ . This algorithm seems to only provide a heuristic solution, but it guaranteed an optimal solution in a finite number of steps, if run to completion. They proposed an algorithm based on Benders' decomposition [50]. The procedure iterates between solving a GAP master problem that assigns vertices to vehicles, and solving a TSPTW to determine the best route for each vehicle. This method has the advantage of

producing a feasible solution, even if not run to completion. Since it repeatedly solves a GAP and a TSPTW, it can benefit from both improvements of these two problems. They also have reported computational results for VRPs ranging from 50 to 199 vertices. This algorithm can also be viewed as a heuristic because it offers an optimal VRP solution at every iteration and is often interrupted during routes execution before optimality can be achieved.

Laporte et al. [51] put forward a more compact formulation for CVRPs and TVRPs, a two-index vehicle flow formulation has the advantage that it is often convenient to impose a lower or an upper bound on the number of vehicles. By means of constraint relaxation algorithm, this model can be solved and have been used on problems containing up to 60 vertices by the same authors.

Garvin et al. [52] first proposed a commodity flow formulation for the CVRP in an oil delivery problem. Baldacci et al. [53] described an integer programming formulation based on a two-commodity flow approach, a lower bound derived from the LP relaxation of the new formulation was improved by adding valid inequalities in a cutting-plane fashion.

Heuristic algorithms for the VRP can often be derived from procedures derived from the TSP. The nearest neighbour algorithm, insertion algorithms and tour improvement procedures can be applied to CVRPs and TVRPs almost without modifications. Following the classification of Laporte et al. [54], heuristic algorithms for the VRP can be divided into two classes: classical heuristics and metaheuristics. The methods of the first class perform a relatively limited exploration of the search space and generally produce good quality solutions within modest computing times. In metaheuristics, a deep exploration of the most promising regions of the solution space can be performed. Combining sophisticated neighbourhood search rules, memory structures, and recombination of solutions, the quality of solutions obtained by these methods is

usually higher than that produced by classical heuristics, but the price is the increase of computing time.

For the classical heuristics, there are two main techniques, one is merging existing routes using a saving criterion, the second is gradually assigning vertices to vehicle routes using an insertion cost. Clarke and Wright [55] first proposed an algorithm based on the saving technique. Gaskell [56], Yellow [57], Paessens [58] have also proposed a number of variants of this method.

Mole and Jameson [59], Christofides et al. [60] proposed two well-known insertion methods for the VRP. However, computational results show that their algorithms are not competitive with the best available methods.

The sweep algorithm can be traced back to the work of Wren [61] and Wren and Holliday [62], but Gillett and Miller [63] made it popular. Feasible clusters are initially formed by rotating a ray centered at the depot. A vehicle route is then obtained for each cluster by solving a TSP. There is an extension of the sweep algorithm called petal algorithms, which is to generate several routes, called petals, and make a final selection by solving a set partitioning problem.

The next is cluster-first route-second algorithms, Fisher and Jaikumar [5] algorithm is the well-known one, which has discussed in the former part. Bramel and Simchi-Levi [64] described a two-phase heuristic in which the seeds are determined by solving a capacitated location problem and the remaining vertices are gradually inserted into their allotted route in a second stage. The authors show that the algorithm is asymptotically optimal (i.e., its solution value tends to the optimum as  $n$  tends to infinity), but its empirical performance is not competitive with that of the best methods.

Improvement heuristics for the VRP operate as a re-optimization procedure, most of them for the TSP can be described in terms of Lin's [65]  $\lambda$ -opt mechanism. A number of edges are removed from the route and the remaining  $\lambda$  segments are reconnected in all possible ways. If any profitable reconnection is identified, it is implemented.

For the metaheuristics, the Tabu Search (TS) algorithm is the main method applied to the VRP in the last two decades. The TS heuristic constructs a sequence of solutions and then executes an improvement step.

The Taburoute algorithm of Gendreau, Hertz and Laporte [66] contains several innovative features. A Generalized insertion (GENI) procedure was used to insert sub solution to one of its nearest neighbours. They also introduced two penalty terms, one measuring over-capacity, the other measuring overduration.

The Taillard [67] TS implementation contains some features of Taburoute, namely random tabu durations and diversification. Rather than executing the insertions with GENI, the algorithm uses standard insertions, which is less time consuming, and feasibility is always maintained. A novel feature of Taillard's algorithm is the decomposition of the main problems into subproblems, which is particularly well suited for parallel implementation as subproblems can be distributed among the various processors. The combination of these strategies yields excellent computational results.

Xu and Kelly [68] proposed a more sophisticated neighbourhood structure, they consider swaps of vertices between two routes, a global repositioning of some vertices into other routes, and local route improvements. The global repositioning strategy solves a network flow model to optimally relocate given numbers of vertices into different routes.

Rego and Roucairol [69] introduced the use of ejection chains to move from one solution to the next. Rochat and Taillard [70] developed Adaptive Memory to enhance a search strategy and got two new best solutions one the 14 standard VRP benchmark instances.

Toth and Vigo [71] introduced a Granular Tabu Search method and yielded excellent results on the VRP. The main idea is that the longer edges of a graph only have a small likelihood of belonging to an optimal solution.

### 3.3 Introduction of the DVRP

The first reference to a DVRP is from Wilson and Colvin [72], who studied a single vehicle Dial-a-ride Problem (DARP), in which the requests are trips from an origin to a destination that appear dynamically. They used insertion heuristic and achieved good performance with low computational cost. Then, Psaraftis [73] introduced the concept of immediate request: need immediate redesign of the current route.

With the development of Global Positioning System (GPS) and Geographic Information Systems (GIS), widespread use of mobile phones, the routing process can be done dynamically, providing more opportunities to improve customer experience and reduce the operational and transportation costs.

Figure 3.1 [74] demonstrates the route execution of a single vehicle D-VRP, which can help understand the word “dynamic”. The initial route planned for current requests is  $(A, B, C, D, E)$  (time  $t_0$ ), while the vehicle executing this route, two new requests appear at time  $t_1$  and the route changes immediately to meet the new demands. Finally, the executed route is  $(A, B, C, D, Y, E, X)$  at time  $t_f$ .

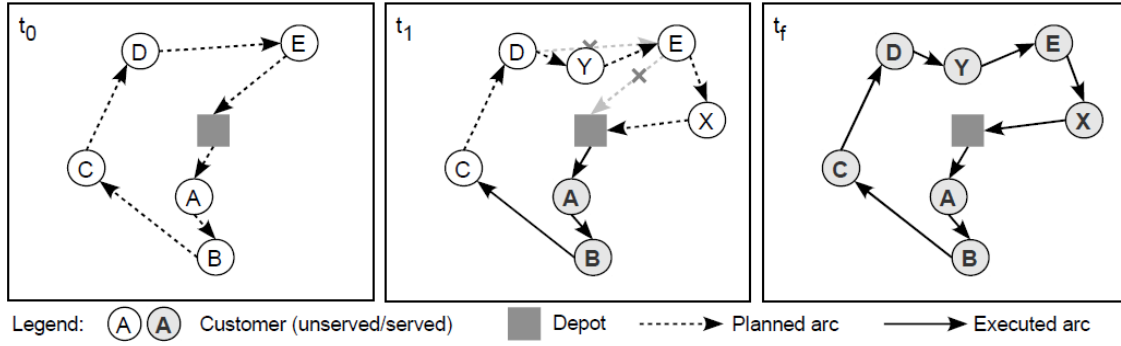


Figure 3.1: Example of dynamic vehicle routing.

Compared with static counterpart, dynamic routing problems involve more degrees of freedom which increase the complexity of decisions and bring new challenges to judge the merit of a planned route.

The first challenge is service guarantee [75], the system may deny a request because it is impossible to service or the cost is too high. The second is that DVRP requires real-time knowledge of the vehicle position and need to communicate quickly with drivers to execute the updated routes. The third one is that its objective function always differs from the static routing [76]. The common objective in the static routing is the minimization of the routing cost, and the dynamic one may introduce new criterions, such as user experience, throughput (number of serviced requests), response time, or revenue maximization. Last but not least, DVRP require making decisions in an online manner, which compromises reactiveness with decision quality as it always need time to search for better decisions.

Different problems can have different levels of dynamism, which can be characterized according to two dimensions [77]: the frequency of changes and the urgency of requests. The

former is the rate at which new information becomes available, while the latter is the time gap between the disclosure of a new request and its expected service time. Based on these two dimensions, three metrics have been proposed to measure the dynamism of a problem.

Lund et al. [78] put forward the degree of dynamism  $\delta$  as the ratio between the number of dynamic requests  $n_d$  and the total number of requests  $n_{tot}$  as follows:

$$\delta = \frac{n_d}{n_{tot}} \quad (3.3.1)$$

Larsen [79] proposed the effective degree of dynamism  $\delta^e$ , which can be expressed as:

$$\delta = \frac{1}{n_{tot}} \sum_{i \in R} \frac{t_i}{T} \quad (3.3.2)$$

$T$  is the length of the planning horizon,  $R$  the set of requests,  $t_i$  the disclosure time of request  $i \in R$ . The disclosure time of requests known beforehand is equal to 0.

Larsen [79] also extended the effective degree of dynamism to problems with time windows to reflect the level of urgency of requests. The effective degree of dynamism measure is extended as follows:

$$\delta_{TW}^e = \frac{1}{n_{tot}} \sum_{i \in R} \left( 1 - \frac{l_i - t_i}{T} \right) \quad (3.3.3)$$

Where  $(l_i - t_i)$  is the reaction time between the disclosure time  $t_i$  and the end of the corresponding time window  $l_i$ , highlighting that longer reaction times mean more flexibility to insert the request into the current routes.

Although the effective degree of dynamism and its variants have proven to capture well the time-related aspects of dynamism, it does not take the geographical distribution of requests or the traveling times between requests.

### 3.4 Literature Review of the DVRP

According to the information quality, DVRP can be classified into two categories: dynamic and deterministic routing problems, and dynamic and stochastic problems.

For the first category, as we discussed in Chapter 1, critical information is revealed over time, so the optimal solution can only be found a-posteriori, and exact approaches only provide an optimal solution for the current state, no guarantee that the solution will be optimal once new requests revealed.

Psaraftis [73] proposed a dynamic programming approach to find the optimal route each time a new request is known on the DARP. The drawback is the curse of dimensionality of dynamic programming, which prevents its application to large instances.

Yang et al. [80] used a rolling horizon approach to solve the real-time truckload pickup and delivery problem (PDP) in which requests arriving dynamically, their approach is based on an LP which is solved whenever a new request arrives.

Chen and Xu [81] designed a dynamic column generation algorithm for the DVRPTW. The main feature of their approach is that it dynamically generates columns for a set-partitioning model, using columns from the previous decision epoch. Their approach yields comparable results with a traditional column generation with no time limit in terms of objective function based on the



Solomon benchmark [82], but running time reduced to 10 seconds, in contrast of several hours by the traditional one.

Since optimal solutions can only be found a-posteriori, most dynamic approaches rely on heuristics to come up with a quick solution for the current state.

Gendreau et al. [83] firstly applied TS to DVRP. They used the adaptive memory to generate initial solutions for a parallel TS. Whenever a new request arrives, it is checked against all the solutions from the adaptive memory to decide whether it should be accepted or rejected. As a consequence, vehicles do not know their next destination until they finish one request.

Bent and Van Hentenryck [84] introduced the Multiple Plan Approach to solve DVRPTW, which is a generalization of approach of Gendreau et al [83]. The advantage of their approach is the solution pool update periodically, which ensures all solutions are coherent with the current state of vehicles and customers.

Benyahia and Potvin [85] used Genetic Algorithm (GA) to model the decision process of a human dispatcher to solve D-PDP. Montemanni et al. [86] developed an Ant Colony System (ACS) to solve DVRP. They divide the day in periods of equal duration as time slices. A request arriving within a time slice is not handled until the end of this time slice. The advantage of this time partition is that similar computational effort is used for each time slice. This discretization is suitable for the requests which are not urgent and can be postponed. Gambardell et al. [87] and Rizzoli et al. [88] also used ACS to solve the similar problems.

For the second category, following the classification of Pillac et al. [74], it can also be divided into two categories: sampling, and stochastic modeling. The former one incorporates

stochastic knowledge by sampling based on realizations or the history data, and then it can be optimized by solving static and deterministic VRP. The latter one captures the stochastic nature of the problem, but is required to compute possibly complex expected values efficiently.

For the sampling, Flatberg et al. [89] adapted the SPIDER commercial solver to use multiple scenarios and a consensus algorithm to solve the DVRP. Pillac et al. [90] implemented an event-driven optimization framework based on Multiple Scenario Approach (MSA) and achieved important improvements for the DVRPSD.

Hvattum et al. [91] proposed the Dynamic Sample Scenario Hedge Heuristic for DVRP, which divides the planning horizon into time intervals. The main feature of their method is using the Branch and Regret Heuristic (BRH) to merge scenarios to build a unique solution.

Ghiani et al. [92] designed an algorithm for the DPDP that only samples the near future to reduce the computational cost. The difference is that no scenario pool used and the selection of the distinguished solution is based on the expected penalty of accommodating the future requests.

Azi et al. [93] designed an Adaptive Large Neighborhood Search (ALNS) to evaluate the opportunity value of an incoming request. Ichoua et al. [94] and Attanasio et al. [95] used TS to solve the DVRPTW and the DPDP, respectively.

For the stochastic modeling, Powell et al. [96] formulated a truckload PDP as a Markov Decision Process (MDP). Thomas and White [97] and Thomas [98] used MDP to solve a VRP in which known customers may ask for service with a known probability. The curse of dimensionality and the simplifying assumptions prevent the wide application in the most real-world cases.

Novoa and Storer [99] proposed an Approximate Dynamic Programming algorithm to dynamically solve the VRPSD. Yang et al. [100] proposed an approach based on LP, which considers opportunity costs on each arc to reflect the expected cost of travelling to isolated areas. As a result, it tends to reject isolated requests and avoids traversing arcs that are far away from potential requests.

Besides, there are other strategies for DVRP, such as waiting strategy, relocation strategy, request buffering and so on, which are not in details in this thesis, because such strategies are not highly related with this thesis.

## Chapter 4

### 4 Dial-a-Ride Problem

The proposed solution framework is based on the use of time slices, and the service time is divided in periods of equal duration. A request arriving during a time slice is not handled until the end of the time bucket, thus the problem solved during a time slice only considers the requests known at its beginning. Within each time slice, the problem can be reduced to be a single vehicle Dial-a-Ride Problem (DARP). This chapter will describe:

- The definition of the DARP,
- The formulation of the DARP,
- The main features of the DARP,
- The review of the single vehicle DARP, and
- The review of the multi-vehicle DARP.

## 4.1 Formulation of the DARP

The Dial-a-Ride Problem (DARP) consists of designing vehicle routes and schedules for  $n$  users who specify pickup and delivery requests between origins and destinations. The objective is to plan a set of  $m$  minimum cost vehicle routes capable of accommodating as many users as possible, under a set of constraints.

Given a complete directed graph  $G(V, E)$  consisting of a set of  $V$  of  $n$  nodes and an arc set  $E$ . Node set  $V$  can be decomposed into three subsets; depot  $v_0$ , pick-up points  $v_l^p$ ,  $l = 1, 2, \dots, m$ , delivery points  $v_l^d$ ,  $l = 1, 2, \dots, m$ . Node  $v_0$  represents a depot that is the starting and ending point of the vehicle. There exist  $m$  customers whose pick-up and delivery point pairs are  $(v_l^p, v_l^d)$ ,  $l = 1, 2, \dots, m$ , where  $v_l^p$  is the pick-up point and  $v_l^d$  is the delivery point of customer  $l$ . We denote the number of nodes by  $n = |V| = 2m + 1$ . Associated with each arc  $(i, j) \in E$ , there is a weight  $D_{ij}$  that represents the distance between node  $i$  and node  $j$ . Our objective is to find a permutation of nodes  $\delta: V \rightarrow \{1, 2, \dots, n\}$  that minimizes the total travel distance

$$\sum_{i=1}^{n-1} D_{\delta^{-1}(i)\delta^{-1}(i+1)} + D_{\delta^{-1}(n)\delta^{-1}(1)}, \quad (4.1.1)$$

And satisfies the following conditions:

1.  $\delta_{(v_0)} = 1$ ,
2.  $\delta_{(v_l^p)} < \delta_{(v_l^d)} \quad \text{for all} \quad l = 1, 2, \dots, m$ .

Constraint 2 states that the pick-up point  $v_l^p$  must be visited before its corresponding delivery point  $v_l^d$ . Note that permutation ... means node  $i$  is the  $j$ -th visiting point of the vehicle.

From a modeling point of view [101], the DARP generalizes a number of vehicle routing problems such as the Pick-up and Delivery Vehicle Routing Problem (PDVRP) and the Vehicle Routing Problem with Time Windows (VRPTW). What makes the DARP different from most such routing problems is the human perspective. When transporting passengers, reducing user inconvenience must be balanced against minimizing operating costs. In addition, vehicle capacity is normally constraining in the DARP whereas it is often redundant in PDVRP applications, particularly those related to the collection and delivery of letters and small parcels.

## 4.2 Main Features of the DARP

Dial-a-ride services may operate according to a static or to a dynamic mode. In the static case, all the requests are known beforehand. In the dynamic mode, the requests are gradually revealed throughout the day and vehicle routes are adjusted in real-time to meet demand. Pure dynamic DARPs rarely exist in practice since a subset of requests is often known in advance.

The studies on the DARP usually assume the availability of a fleet of  $m$  homogeneous vehicles based at a single depot. Even though this hypothesis often reflects reality and can serve as a sound base for the design of models and algorithms, it is significant to realize that different situations exist in practice. There may be several depots, especially in wide geographical areas, and the fleet of vehicles is sometimes heterogeneous. The functions of vehicles may vary, some of them are designed to provide services to the disabled, others may only cater to ambulatory passengers and some of them may accommodate both types of services. The main consideration in some problems is to first determine a fleet size and composition capable of satisfying all demands while in others, the objective is to maximize the number of requests that can be served with a fixed size of vehicles. Some systems routinely turn down several requests each day. A

compromise consists of serving some of the demands with a core vehicle fleet and using extra vehicles if necessary.

Quality of service criteria include route duration, route length, customer waiting time, customer ride time (i.e., total time spent in vehicles), and the difference between actual and desired drop-off times. Some of these criteria may be treated as constraints or as part of the objective function. There is a common trend in DARP models to let users impose a time window on both their departure and arrival times. However, if the time windows are too narrow, there may appear the deny of requests in practice, which absolutely decreases the quality of service.

### **4.3 Review of the Single Vehicle DARP**

DARP is NP-complete [102], a polynomial time algorithm most unlikely exists for solving the DARP exactly. Consequently, for practical problems with several hundreds of nodes they must use an approximate algorithm that runs in low polynomial time order.

Kalantari et. al. [103] provided a branch and bound method for the precedence constrained traveling salesman problem (PCTSP) based on the branch and bound algorithm for the TSP by Little et. al [104], and could solve the 30-node problems with the asymmetric distance matrix. Suzuki and Nomura [105] also developed a branch and bound algorithm for the PCTSP using bounds derived from the arborescence problem or the shortest path problem, and could solve 30-node problems with the sparse distance matrix. In their model, nodes are permitted to be visited more than once and the starting and ending points are distinct, but this problem can be reduced to the ordinary PCTSP using a simple transformation [106]. Psaraftis [73] developed an exact

solution method using a dynamic programming algorithm for the DARP with additional constraints whose time complexity is  $O(n^2 3^n)$ , and could solve 20-node problems.

Several heuristic algorithms have been proposed for the DARP (Psaraftis [107], [108], Jaw et. al. [109], Stein [110]). Psaraftis proposed the minimum spanning tree (MST) heuristic and local search (k-opt) procedures specialized to the DARP.

The MST heuristic requires  $O(n^2)$  computations, while the k-opt procedure requires  $O(n^k)$  computations under the appropriate implementation described in [108]. Since the computational complexity of  $k$ -opt procedure grows in exponential order of  $k$ , Psaraftis recommended  $k = 2, 3$  (2-opt, 3-opt) in practice.

Stein [110], [111] provided an algorithm for the Euclidean DARP based on the Karp's algorithm [112] for the TSP, and proved the algorithm produces asymptotically optimal solutions when  $l$  approaches infinite.

Daganzo [113] developed analytical models to evaluate the performance of the DARP. Jaw et. al. [109] provided an insertion heuristic for the multiple vehicle DARP with several additional constraints such as the time window and service quality constraints. Their algorithm builds tours of multiple vehicles through sequential insertion of customers in a dynamically changing environment.

M. Kubo and H. Kasugai [106] proposed several new heuristic algorithms for the DARP, and made comparisons among four classes of algorithms. The first class are based on the sequential insertion of nodes introduced in [109]. The second is an extended version of the classical nearest neighbor method. The third one uses the spacefilling curve heuristic [114] and can be seen as a



simplified version of the Stein's asymptotically optimal heuristic algorithm [110]. The fourth are local improvement procedures generalized of the Or-opt procedure [115] for the TSP.

Numerical experiments are implemented to compare the performance of the approximate algorithms of these four classes mentioned above. The experiments are on  $100 * 100$  grids and the distance matrix is generated according to two dimensional Euclidean distances between nodes. The maximum is  $n = 101$ ,  $n$  is the number of nodes. They used Modified Pairing Insertion (MPI) as the benchmark to compare the different methods with various problem sizes, from  $n = 11$  to  $n = 101$ . Computational time and local search heuristics are also made comparisons based on the same problem sizes.

The conclusion summarized that MPI performs best and could find a dial-a-ride tour within about 6% optimality. The local search procedures with random starting solutions produce worse solutions and require much more computational time than the MPI method. Whereas starting with random solutions, Psaraftis's 3-opt procedure performs best. Whereas when starting with solutions using MPI method, the composite Or-opt procedure 1 performs best. Both 3-opt and composite Or-opt procedures give a dial-a-ride tour within about 1% of optimality with high regularity starting with solutions of the MPI method. The composite Or-opt 1 is faster than the composite Or-opt 2 and recommended the latter.

#### **4.4 Review of the Multi-Vehicle DARP**

Jaw et al. (1986) [109] firstly proposed a heuristic method for the multi-vehicle static DARP. The model they considered imposes time windows on the pick-up times of inbound requests and on the drop-off times of outbound requests. For every request, a maximum ride time, denoted as a linear

function of the direct ride time is imposed. And all the vehicles are not allowed to be idle when carrying passengers. A non-linear objective function combining several types of disutility is used to assess the quality of solutions. The heuristic selects users in order of earliest feasible pick-up time and gradually inserts them into vehicle routes so as to yield the least possible increase of the objective function. Their algorithm was tested on artificial instances involving 250 users and on a real data set with 2617 users and 28 vehicles.

There is a commonly used technique in such problems consisting of defining clusters of users to be served by the same vehicle, prior to the routing phase. Bodin and Sexton (1986) [116] used this idea to construct the clusters by grouping users who are close together in a combined space and time dimension before applying to each cluster the single vehicle algorithm of Sexton and Bodin (1985a, b) [117] [118] and making swaps between the clusters. Numerical results are presented on two instances extracted from a Baltimore database and containing approximately 85 users each.

Dumas et al. (1989a, b) [119] [120] later improved upon this two-phase approach by creating so-called “mini-cluster” of users, i.e., groups of users to be served within the same area at approximately the same time. These mini-clusters are then optimally combined to form feasible vehicle routes, using a column generation technique. Finally, each vehicle route is reoptimized by means of the single vehicle algorithm of Desrosiers et al. (1986) [121], and a scheduling step is executed. They have successfully solved instances derived from real-life data taken from three Canadian cities: Montreal, Sherbrooke and Toronto. Instances with up to 200 users are easily solved, while larger instances require the use of a spatial and temporal decomposition technique.

After that, Desrosiers et al. (1991) [122] later improved the mini-clustering phase and presented results on a dataset comprising almost 3000 users. Finally, Ioachim et al. (1995) [123] displayed that there was an advantage to resorting to an optimization technique to construct the clusters in terms of solution quality.

A real-life problem arising in Bologna was tackled by Toth and Vigo (1996) [124]. Users specify requests with a time window on their origin or destination. A limit proportional to direct distance is imposed on the ride time. Transportation is supplied by a fleet of capacitated minibuses and special cars. Taxis can also be used but since these are not the best choice of transportation for the disabled people, a penalty is imposed on their use. The objective is to minimize the total cost of service. They have developed a heuristic method consisting of first assigning requests to routes by means of a parallel insertion procedure, and then performing intra-route and inter-route exchanges. Tests performed on instances involving between 276 and 312 requests show significant improvements with respect to the previous hand-made solutions. Further improvements were later made by Toth and Vigo (1997) [125] through the execution of a tabu thresholding post-optimization phase after the parallel insertion procedure.

Borndörfer et al. (1997) [126] also uses a two-phase approach in which clusters of users are first constructed and then grouped together to form feasible vehicle routes. A cluster is defined as a “maximal subtour such that the vehicle is never empty”. Its two end-points correspond to the pick-up of the first user and the drop-off of the last user, respectively. In the first phase, a large set of good clusters is constructed and a set partitioning problem is then solved to select a subset of clusters serving each user exactly once. In the second phase, feasible routes are enumerated by combining clusters and a second set partitioning problem is solved to select the best set of routes

covering each cluster exactly once. Both set partitioning problems are solved by a branch-and-cut algorithm. On real-life instances, the algorithm cannot always be run to completion so that it must stop prematurely with the best-known solution. It was applied to instances including between 859 and 1771 transportation requests per day in Berlin.

Wolfler Calvo and Colomi (2002) [127] have devised a heuristic method for a version of the DARP in which the number of available vehicles is fixed and windows are imposed on both pick-up and drop-off times. A hierarchical objective function is utilized: the algorithm first attempts to service as many users as possible and then minimizes user inconvenience expressed as the sum of waiting time and excess ride time. The heuristic first constructs a set of  $m$  routes and a number of subtours by solving an assignment problem. A routing phase is then performed to insert the subtours into the  $m$  routes and to re-sequence the vertices within the routes. Tests were carried out on instances involving between 10 and 180 users.

Cordeau and Laporte (2002) [128] developed a new heuristic on the multi-vehicle static DARP. They applied tabu search to the problem. In their model, users specify a window on the arrival time of their outbound trip and on the departure time of their inbound trip, and a maximum ride time is associated with each user. It can either be the same for all users, or computed by using a maximum deviation factor from the most direct ride time of each particular user. Capacity and maximum route length constraints are imposed on the vehicles. The search algorithm iteratively removes a transportation request and reinserts it into another route. Intermediate infeasible solutions are allowed through the use of a penalized objective function. The minimum duration schedule associated with each candidate solution is computed. The algorithm was tested on randomly generated instances ( $24 \leq n \leq 144$ ) and on six datasets ( $n = 200$  and  $295$ ) provided

by a Danish transporter. With respect to alternative algorithms such as column generation and branch and cut, tabu search can easily accommodate a large variety of constraints and objectives, even if these are non-linear.

Relatively little research on the multi-vehicle dynamic DARP is reported in the scientific literature [101]. Madsen et al. (1995) [129] have solved a real-life problem involving services to elderly and disabled people in Copenhagen. Users may specify a desired pick-up or drop-off time window, but not both. Vehicles of several types are used to provide service, not all of which are available at all times. Requests arrive dynamically throughout the day, and the speed of vehicles is variable and vehicles may become unavailable due to breakdowns. The authors have developed an insertion algorithm, named REBUS, based on the procedure previously developed by Jaw et al. (1986) [109]. New requests are dynamically inserted in vehicle routes taking into account their difficulty of insertion into an existing route. The algorithm was tested on a 300-customer, 24-vehicle problem. They also reported that their algorithm was capable of generating good quality solutions within very short computing times.

Borndörfer et al. (1997) [126] noted that the distinction between static and dynamic DARPs is often blurred in practice since requests are usually cancelled and, consequently, transporters may allow the introduction of new requests in a solution designed for a static problem.

As mentioned before, dynamic DARPs rarely exist in a pure form since a number of requests are often known when planning starts. The difficulty is then to design seed vehicle routes for these requests with sufficient slack to accommodate future dynamic demands.

## Chapter 5

### 5 Algorithms

This chapter is mainly about the algorithms used in the solution framework of this thesis. The contents of this chapter will include:

- The algorithm (Group Role Assignment) applied in phase I,
- The four different routing algorithms applied in phase II,
- The process and update of the data of requests,
- The overall program flow chart, and
- The platform of simulation.

## 5.1 The Proposed Algorithms

The proposed solution framework is based on the use of time slices, and the service time is divided into periods of equal duration. A request arriving during a time slice is not handled until the end of the time bucket, thus the problem solved during a time slice only considers the requests known at its beginning.

In each time slice, a two-phase procedure (cluster-first route-second strategy) is used for solving the Vehicle Routing Problem (VRP). Compared with the route-first cluster-second strategy, the proposed method is based on the Location-based heuristic (LBH) and take the priorities of all the customers into account. As the time running, the new requests are combined with the remaining ones whose priorities are increased at the same time. As a consequence, the passengers can get a better user experience because of the evolution of their priorities, i.e., the passengers with higher priorities are more likely to be served firstly.

### 5.1.1 Phase I – Assign passengers to buses

As earlier mentioned, Group Role Assignment Problem (GRAP) [6] is the method we used to assign passengers to different buses. Since the original requests of passengers only contain the information of start locations and destinations and time-consuming, the raw data need to be pre-processed.

There are three attributes used to decide the priorities of all the passengers:

- 1 . the distance  $d_1$  between the start location and the locations of all the buses;
- 2 . the distance  $d_2$  every passenger need to travel (from one's origin to one's destination);
- 3 . the time-consuming  $t$  of every passenger.

For the first two attributes, both distance is the Manhattan distance, using the following equation to pre-process the data:

$$x_1 \text{ or } x_2 = \frac{d_{max} - d}{d_{max}} \quad (5.1.1)$$

Where  $d_{max}$  is the maximum distance in the graph,  $d$  is  $d_1$  or  $d_2$  with respect to  $x_1$  or  $x_2$ .

For the third attribute, using the following equation to get  $x_3$ :

$$x_3 = \frac{t}{t_{tor}} \quad (5.1.2)$$

where  $t_{tor}$  is the maximum tolerance time for all the passengers.

After the pre-processing, the input data for GRAP can be achieved using the equation:

$$input = c_1x_1 + c_2x_2 + c_3x_3 \quad (5.1.3)$$

Where  $c_1, c_2, c_3$  are the coefficients of  $x_1, x_2, x_3$ , respectively, and the constraint  $c_1 + c_2 + c_3 = 1$  can make sure all the input to be uniform, and the importance of three attributes can be adjusted by changing the coefficients respectively. As the start location of every request is always not the same as the destination, the input can be guaranteed greater than 0.

Besides, the input value obtained from the pre-processing reflects the priorities of the respective passenger because the three attributes of every passenger are reflected on the same scales.

Actually, following the definitions of GRAP in [6], the input matrix is the qualification matrix  $Q$ . After putting the input matrix into the GRAP, the assignment can be obtained in which



every passenger is assigned to a bus exclusively. Then the requests for every bus to be handled can be formed from the assignment.

### 5.1.2 Phase II – Design the routes for buses

In phase II, the task is to design the routes for all the buses. Compared with the route-first cluster-second strategy [3], the complexity of route design is alleviated largely because the requests are partitioned into different buses.

Traditionally, designing the route for a vehicle need to solve the well-known Travelling Salesman Problem (TSP), which is also used in the method of Fisher and Jaikumar [5]. Given a set of cities and distance between every pair of cities, the TSP is to find the shortest possible route that visits every city exactly once and returns to the starting point. It is a famous NP hard problem.

For transporting passengers, the Dial-a-Ride problem (DARP) must be solved for every vehicle within each time slice. The Dial-a-Ride Problem (DARP) consists of designing vehicle routes and schedules for  $n$  users who specify pickup and delivery requests between origins and destinations. The objective is to plan a set of  $m$  minimum cost vehicle routes capable of accommodating as many users as possible, under a set of constraints. As mentioned in Chapter 4, DARP is NP-complete [102].

Despite the difficulty of the DARP, the route obtained from solving DARP is the shortest length solution, which is not the major objective of ours, because the main objective of the transit system is to use the minimum time to transport passengers to their destinations with the limit of transportation power. In addition, the priorities obtained from phase I are completely futile if using the shortest route of DARP.

In contrast of the method of Fish and Jaikumar [5], four different routing methods are designed based on the different perspectives.

The first one is using an insertion heuristic, which is based on the algorithm proposed by Jaw et al. [109]. The strategy is to consider the pickup at first, insert all the destinations to the pickup route one by one according to the priorities. The route generation of every vehicle can be demonstrated as below.

-----**The insertion heuristic routing algorithm based on Jaw et al. [109]**-----

**Step 1:** Based on the priorities and time-consuming obtained from phase I, sort the information of passengers in a descending order;

**Step 2:** Put all the start locations of the sorted passengers from step 1 into the route;

**Step 3:** Insert the destinations of all the passengers into the route one by one based on the order achieved from step 1, as the bus need to pick-up and delivery simultaneously, the successful insertion of destination is always below its origin, because one passenger need to be picked up first and then delivered to one's own destination to complete a service.

**Step 4:** Find the position of first drop-off, collect all the pick-up locations of the passengers before the first drop-off, using a local search method to compare the cost of swap one's pick-up location with another's, if the swap makes the route shorter than the optimal one, update the optimal route, otherwise no action. Continue Step 4 until no improved solution can be found.

**Step 5:** Find the position of last passenger pick-up, collect all the destinations of the passengers after it, using a local search method to compare the cost of swap one's destination with

another's destination, if the swap makes the route shorter than the optimal one, update the optimal route, otherwise no action. Continue Step 5 until no improved solution can be found.

---

The second routing method is based on the composite Pairing Or-opt method [106], which uses a Pairing Or-opt exchange procedure to get the route.

The algorithm of the Pairing Or-opt method is demonstrated as below.

---

**-----The Pairing Or-opt routing algorithm-----**

**Step 1:** Get an initial feasible route.

**Step 2:** Delete a pair of pick-up and destination points from the current tour, and insert them into the subtour to satisfy the precedence constraints and to minimize the increase of the distance. Continue Step 2 until no improved solution can be found.

---

Based on the algorithm above, the second routing algorithm can be demonstrated as below.

---

**-----The Composite Pairing Or-opt routing algorithm-----**

**Step 1:** Generate the initial route based on the priorities and time-consuming obtained in phase I, put every destination after its origin, set it as the optimal route;

**Step 2:** Using a local search method to compare the cost of swap one's destination with another's origin and destination respectively, choose the shorter one and compare with the optimal

route, if it is shorter than the optimal one, update the optimal route, else no action. Continue Step 2 until no improved solution can be found.

**Step 3:** Find the position of first drop-off, collect all the pick-up locations of the passengers before the first drop-off, using a local search method to compare the cost of swap one's pick-up location with another's, if the swap makes the route shorter than the optimal one, update the optimal route, otherwise no action. Continue Step 3 until no improved solution can be found.

**Step 4:** Find the position of last passenger pick-up, collect all the destinations of the passengers after it, using a local search method to compare the cost of swap one's destination with another's destination, if the swap makes the route shorter than the optimal one, update the optimal route, otherwise no action. Continue Step 4 until no improved solution can be found.

---

We observe that the objective of GRAP is to maximize the performance of all the passengers, which always assign passengers with both high and relatively low priorities to a vehicle. And in each time slice, the distance of a vehicle can travel is limited, as a result, even though the optimal route we can get in each time slice, the vehicle cannot finish it completely and with the new requests combined in the next iteration, the pressure of both routing and time-consuming of the passengers in the vehicle will add up further more. Consequently, completing as many requests as possible in each iteration can not only alleviate the pressure of both capacity and the routing process, but also reduce the time of passengers staying on the vehicles. The third and fourth routing algorithms can be seen as improved methods based on the first and second routing methods respectively. The procedure is as below.

-----**The Modified Insertion Heuristic routing algorithm**-----

**Step 1:** Based on the priorities and time-consuming obtained from phase I, sort the information of passengers in a descending order;

**Step 2:** Put all the start locations of the sorted passengers from step 1 into the route;

**Step 3:** Insert the destinations of all the passengers into the route one by one based on the order achieved from step 1, as the bus need to pick-up and delivery simultaneously, the successful insertion of destination is always below its origin, because one passenger need to be picked up first and then delivered to one's own destination to complete a service.

**Step 4:** Find the position of first drop-off, collect all the pick-up locations of the passengers before the first drop-off, using a local search method to compare the cost of swap one's pick-up location with another's, if the swap makes the route shorter than the optimal one, update the optimal route, otherwise no action. Continue Step 4 until no improved solution can be found.

**Step 5:** Find the position of last passenger pick-up, collect all the destinations of the passengers after it, using a local search method to compare the cost of swap one's destination with another's destination, if the swap makes the route shorter than the optimal one, update the optimal route, otherwise no action. Continue Step 5 until no improved solution can be found.

**Step 6:** Based on the route of step 5, partition the passengers into two groups, the first one is the passengers can get the service in the current time slice, the other one is the passengers cannot get the service in this iteration;

**Step 7:** Using the routing method of step 1-5 to redesign the routes for both two groups separately.

**Step 8:** Combine both routes into one as the final route.

-----**The Modified Composite Pairing Or-opt routing algorithm**-----

**Step 1:** Generate the initial route based on the priorities and time-consuming obtained in phase I, put every destination after its origin, set it as the optimal route;

**Step 2:** Using a local search method to compare the cost of swap one's destination with another's origin and destination respectively, choose the shorter one and compare with the optimal route, if it is shorter than the optimal one, update the optimal route, else no action. Continue Step 2 until no improved solution can be found.

**Step 3:** Find the position of first drop-off, collect all the pick-up locations of the passengers before the first drop-off, using a local search method to compare the cost of swap one's pick-up location with another's, if the swap makes the route shorter than the optimal one, update the optimal route, otherwise no action. Continue Step 3 until no improved solution can be found.

**Step 4:** Find the position of last passenger pick-up, collect all the destinations of the passengers after it, using a local search method to compare the cost of swap one's destination with another's destination, if the swap makes the route shorter than the optimal one, update the optimal route, otherwise no action. Continue Step 4 until no improved solution can be found.

**Step 5:** Based on the route of step 4, partition the passengers into two groups, the first one is the passengers can get the service in the current time slice, the other one is the passengers cannot get the service in this iteration;

**Step 6:** Using the routing method of step 1-4 to redesign the routes for both two groups separately.

**Step 7:** Combine both routes into one as the final route.

---

### 5.1.3      **Execute the routes and update the information**

The remaining work is to execute the routes of all the buses and update the information of all the passengers and the bus locations, the information update of passengers includes removing the arrived passengers from the pending matrix, the information of not arrived passengers update and combine with the new requests into the next iteration. The overall program flow chart is as below:

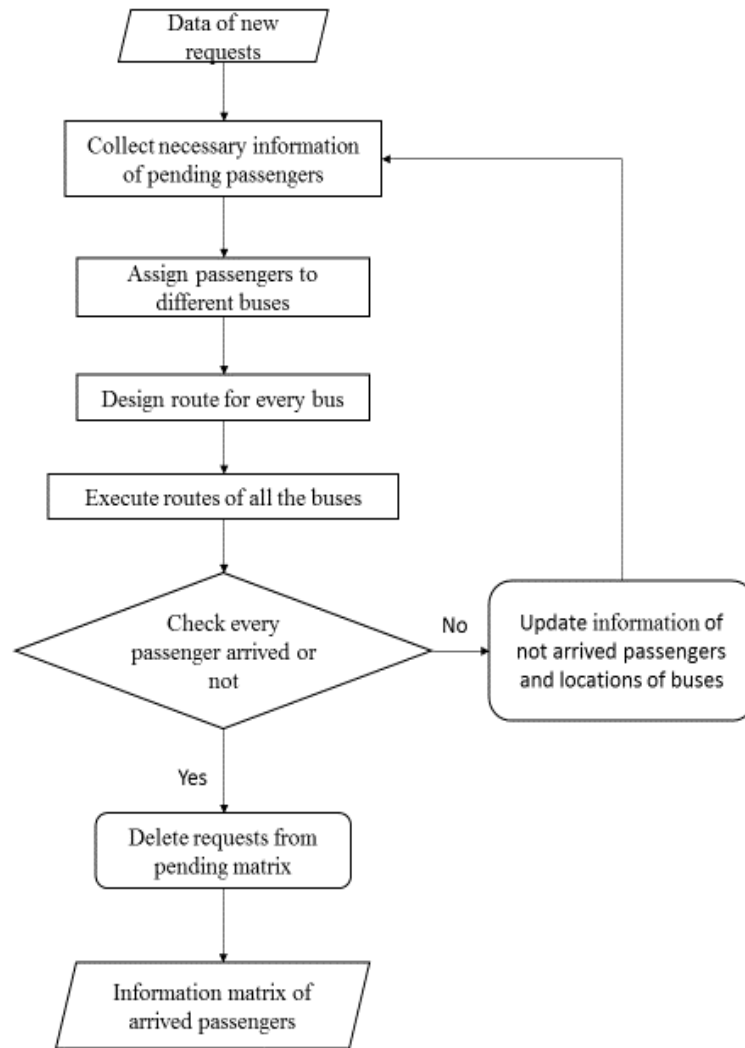


Figure 5.1: The overall program flow chart.

## 5.2 Platform of simulation

The simulation is performed on a PC running Intel Core M-5Y10C processor clocked at 998Mhz with dual cores and 8 GB RAM running Windows 10 Pro edition.

The software is MATLAB R2016a.



To make the readers easy to understand the key mechanism of our algorithm, Appendix III has provided an example of 2 vehicles to serve 10 passengers in two continuous time slices.

The simulation results of high intensity of requests is presented in Chapter 6.

## Chapter 6

### 6 Simulation Results and Analysis

This chapter is to present the simulation results and analyze the performance of the results of different simulations. The contents will include:

- Instance generation,
- Simulation results of different coefficients,
- Simulation results of different routing algorithms,
- The equal pressure test,
- The differential pressure test, and
- The qualitative comparison of different methods in this field.

## 6.1 Instance generation

Our simulation takes North Bay as the object. Based on the map of North Bay transit system, a  $25 \times 20$  grid is put to cover the majority area of the whole city, the unit length is 500 meters. According to the actual distribution of bus stops, 52 stops on the grid are chosen to cover the most service area of the transit system. The requests are generated from these 52 stops randomly. The speed of vehicles is set to a fixed value, 60 km/h. Assume all the buses can run without stop and ignore the factors, such as time cost of pickup and drop-off, the computing time of our approach, the time consuming of transfer, the break time for drivers etc. The time slice is set to 15 minutes, so within each time slice, the vehicles can travel 30 units. It also means that it cost 0.5 minute to travel one unit in the grid for all the vehicles. Our experiments are also based on the information to estimate the time cost of every passenger.

The simulation is based on the length of the grid, which is the Manhattan distance, not the Euclidean distance, because the practical road condition is complicated, using Euclidean distance cannot get a more accurate time-consuming result close to the simulation in the practical environment. As the major roads are designed horizontal or directional, the length of the grid is closer to the practical circumstances.

## 6.2 Simulation results of different coefficients

To get a better simulation result, the first experiment is made to choose the coefficients for the three attributes with the minimum average time-consuming. The time slice is set to 15 minutes. The number of vehicles is 13, same as the North Bay transit system. This simulation is using the

routing method II in Chapter 5. The capacity of all the vehicles is 20. The data of both passengers and buses will be attached in the Appendix I. The result is in table 6.1.

Table 6.1: The comparison of using different coefficient datasets.

Total number of passengers	Number of passengers of each load	Coefficient array	Minimum time cost(min)	Maximum time cost(min)	Average time cost(min)	CPU running times(s)
400	100	[1/3,1/3,1/3]	1.5	239.5	108.3	4.419
400	100	[0.4,0.3,0.3]	3.0	236.5	102.0	3.373
400	100	[0.3,0.4,0.3]	3.0	244.5	109.0	4.377
400	100	[0.3,0.3,0.4]	1.5	276.0	128.1	5.335
400	100	[0.5,0.1,0.4]	3.0	264.5	125.1	5.912
400	100	[0.5,0.4,0.1]	2.0	239.5	105.3	4.640
400	100	[0.1,0.4,0.5]	2.0	242.5	103.7	5.170
400	100	[0.1,0.5,0.4]	2.5	250.0	106.5	4.676
400	100	[0.4,0.5,0.1]	3.0	252.0	115.2	5.504
400	100	[0.4,0.1,0.5]	3.5	268.5	128.7	6.723

According to the numerical experiment above, when using [0.4, 0.3, 0.3] as the coefficients, the average time-consuming is smaller than the others, so [0.4, 0.3, 0.3] will be used as the coefficients in the next related experiments.

### 6.3 Simulation results of different routing algorithms

To test the effectiveness of different routing methods, this experiment will also load 100 passengers in each time slice (15 min) and compare the performance of all the four routing methods. This test contains 400 passengers in total, and the total route length of the requests of all the passengers is 2487.5 km.

I: Insertion heuristic routing method based on Jaw et al. [109].

II: Composite pairing Or-opt routing method.

III: Modified insertion heuristic routing method based on time slice.

IV: Modified composite pairing Or-opt routing method based on time slice.

Table 6.2: The comparison of using different routing algorithms.

Routing method	Maximum time cost(min)	Average time cost(min)	Actual route length of vehicles (km)	CPU running times(s)
I	144.5	58.9	1896.5	2.032
II	236.5	102.0	2988.5	3.909
III	144.0	56.4	1831.0	2.613
IV	219.0	85.5	2777.5	5.041

From the numerical results of Table 6.2, routing method III outperforms the other three from every aspect compared in this test. However, since the strategy of routing method III is pick-up first, from the result of the actual route, it need vehicles with high capacity. Routing method IV is better than routing method II from the maximum time cost and average time cost. As a result, routing algorithm III and IV will be used in the remaining experiments. The capacities of vehicles for routing algorithm III and IV are 24 and 20, respectively.

## 6.4 Equal pressure test

To test the maximum capability of our system, I design this equal-pressure test. The numerical results are in Table 6.3. As the number of vehicles is limited to 13, the minimum number of passengers of each load is set to 13.

Table 6.3: The numerical results of equal-pressure test using routing algorithm III.

Total number of passengers	Number of passengers of each load	Minimum time cost(min)	Maximum time cost(min)	Average time cost(min)	CPU running times(s)
400	100	1.5	144.0	56.4	2.777
400	80	2.0	126.5	52.0	2.525
400	60	1.5	135.0	42.6	1.702
400	40	1.0	126.0	31.0	1.677
400	20	0.5	56.5	13.6	1.432
400	13	1.0	50.0	9.9	1.405

Table 6.4: The numerical results of equal-pressure test using routing algorithm IV.

Total number of passengers	Number of passengers of each load	Minimum time cost(min)	Maximum time cost(min)	Average time cost(min)	CPU running times(s)
400	100	2.5	219.0	85.5	5.200
400	80	1.5	209.5	73.6	3.891
400	60	1.5	213.5	66.1	3.498
400	40	2.5	154.5	39.2	1.874
400	20	0.5	68.5	12.3	1.305
400	13	1.0	42.0	9.5	1.067

From the results of Table 6.3 and 6.4, it is apparent that with the decrease of the number of passengers in each load, the average time-consuming is reducing rapidly and the maximum time-consuming is also decreasing in general. The CPU running time is a reflection of the load of our system.

Compared with the results of both routing algorithms, when the number of passengers of each load is high, more specifically from 100 to 40, routing algorithm III has better results. When it is relatively low, from 20 to 13, routing algorithm IV has more benefits.

## 6.5 Differential pressure test

To make our program more practical for the application in the real world, the differential-pressure experiment is designed to test the practicability and capability. This test totally contains 15 vehicles

and 150 passengers, which are the first 150 requests of 400 passengers used in the above experiments, and they will be put into our system time slice by slice with the numbers of 50, 40, 30, 20, 10 respectively. The results are in Table 6.5 and 6.6. Both algorithms can use vehicles with capacity of 5. The detailed result of time-consuming of every passenger will be attached in Appendix II.

Table 6.5: The numerical results of differential-pressure test using routing algorithm III.

Total number of passengers	Numbers of passengers of every load	Minimum time cost(min)	Maximum time cost(min)	Average time cost(min)	CPU running times(s)
150	50, 40, 30, 20, 10	2.0	67.0	22.0	1.319

Table 6.6: The numerical results of differential-pressure test using routing algorithm IV.

Total number of passengers	Numbers of passengers of every load	Minimum time cost(min)	Maximum time cost(min)	Average time cost(min)	CPU running times(s)
150	50, 40, 30, 20, 10	1.0	75.0	20.7	0.986

From the test result, the maximum time cost has been limited to 67.0 and 75.0 minutes with using the different routing algorithms respectively. And the average time cost has been reduced to 22.0 and 20.7 minutes respectively. One more important thing to mention is that the time consuming is not only the time cost in the vehicles, it is the sum of the waiting time and the time spending in the vehicles. For the waiting time, our system has the ability to give an estimated waiting time, consequently, passengers no longer need to spend the whole time listed in Appendix II to get their ridership, they just need to arrive their pickup location before the pickup time due.

From the results of the routes of all the vehicles, when using routing algorithm III, our system need vehicles with high capacity. When using routing algorithm IV, our system can use small cars with the capacity of 4 (excluding the driver) to serve the passengers. Furthermore, with the use of small cars, the cost of vehicles and drivers will be reduced in some extent. If the private cars can be used in the system, like DiDi carpool service, the budget of the whole system will be decreased more than expected.

To identify the practicability and avoid the extreme case in this differential pressure test, we have made 20 more datasets for our experiment, each dataset has 150 requests and also are put into the system time slice by slice with the numbers of 50, 40, 30, 20, 10 respectively. The start coordinates are the same with test in Table 6.5 and 6.6, and the detailed data is in Appendix II. The results are in Table 6.7 and 6.8.

Table 6.7: The numerical results of 20 datasets with differential-pressure test using routing algorithm III.

Index of dataset	Maximum time cost(min)	Average time cost(min)	CPU running times(s)
1	45.0	14.8	0.997
2	54.0	15.6	0.906
3	73.5	15.1	0.917
4	52.0	16.7	0.938
5	63.5	16.2	0.983
6	58.5	16.9	1.061
7	65.5	18.1	0.959
8	60.5	16.4	0.946



9	59.5	19.3	0.994
10	47.0	15.8	0.934
11	60.0	16.5	0.965
12	65.5	18.7	1.019
13	58.0	15.6	0.912
14	75.0	16.5	0.918
15	60.0	16.2	0.964
16	56.0	15.1	0.899
17	39.5	14.4	0.905
18	62.0	15.9	0.915
19	67.5	18.2	0.955
20	68.0	17.8	0.991

Table 6.8: The numerical results of 20 datasets with differential-pressure test using routing algorithm IV.

Index of dataset	Maximum time cost(min)	Average time cost(min)	CPU running times(s)
1	56.0	14.4	1.060
2	52.5	14.2	0.922
3	63.0	15.1	0.903
4	58.5	16.1	0.939
5	47.5	14.8	0.920
6	68.0	17.9	1.005
7	62.0	16.1	0.828

8	57.5	15.9	0.919
9	69.5	16.6	0.946
10	64.0	16.8	0.967
11	49.0	14.9	0.998
12	66.5	20.5	0.989
13	46.5	15.4	0.979
14	77.5	17.6	1.166
15	51.0	14.1	1.129
16	57.0	14.8	1.081
17	71.5	16.2	1.190
18	55.5	17.0	1.123
19	44.0	15.7	1.131
20	58.5	16.3	1.158

## 6.6 Qualitative Comparison

There are some related studies in this area from different perspectives, such as Madsen et. al. [129] presented an algorithm for dynamic DARP for the transportation of elderly and handicapped people, Horn [130] developed a software for demand-responsive passenger services such as taxis and variable route buses, Beaudry et. al. [131] developed a two-phase algorithm for solving a complex dynamic DARP arising in the transportation of patients in hospitals, Cordeau et. al. [132] presented the dial-a-flight problem to model and optimize on-demand air charter services etc.

As the related studies are based on different perspectives and have different objectives, the data sets used in the experiments differ, making a quantitative comparison is not likely to come true. Consequently, making a qualitative comparison among the main algorithms can give more valuable and meaningful information and performance evaluation. The data of our algorithm used in this comparison is from Table 6.6. The detailed comparison result is in Table 6.9.

Table 6.9: The qualitative comparison results of different algorithms.

Authors	Algorithms	Number of requests	Number of vehicles	Time duration (h)	Number of Services per vehicle per hour	Service Guarantee (Yes or No)	Capacity constraint considered (Yes or No)	Idle of vehicles (Yes or No)
Madsen et. al.	Insertion algorithm based on Jaw et.al. [110]	300	24	Not mentioned	Not available	Yes	Yes	No
Horn	Minimum cost insertion based on local search	4200	220	24	0.80	Yes	Yes	Yes
Attanasio et. al. [133]	Sequential insertion algorithm based on tabu search	144	10	Not mentioned	Not available	No	Yes	Yes
Caramia et. al. [134]	Neighborhood search and dynamic programming routing	400	30	8	1.67	No	Yes	Yes
The proposed one	Group Role Assignment and Modified composite pairing Or-opt routing	150	15	1.38	7.23	Yes	Yes	No

From the comparison of Table 6.5, we can see that our solution is better on the efficiency of using vehicles to service passengers compared with Horn and Caramia et. al. Compared with the algorithms of Attanasio et. al. and Caramia et. al., the other three algorithms do not deny passengers' requests to present a significant indicator of user satisfaction. All the algorithms proposed have taken the capacity constraint of vehicles into consideration. Only the algorithms of Madsen et. al. and ours will not let the vehicles have slack time, which is another criterion for the exploitation of the transportation power of all the vehicles.

If put our algorithm into operation in the real world, we can provide the function to let the passengers to choose whether accept or reject the route designed by the system to further improve the user experience and satisfaction.

## Chapter 7

### 7 Conclusion

In this thesis, a new solution framework has been proposed to design a dynamic ridesharing system to provide services to the public. From the view of the overall program, the system is dynamic and the routes of all the vehicles are changing with the running of time slices. From the standpoint of each time slice, with the use of this kind of set partitioning strategy, the problem has been reduced to a static Dial-a-ride problem (DARP) in each time slice. Consequently, not only the complexity of routing can be reduced largely, but also the problems of the pure form of dynamic DARPs can be avoided, such as the confusion of drivers, the difficulty of redesigning the route etc.

With the use of Group Role Assignment Problem (GRAP) [6], the work of assigning passengers to different vehicles not only enjoys the benefit of Location-based heuristic (LBH), which can be asymptotically optimal and has been proven by Bramel and Simchi-Levi [4], but also can take the total time spending (the sum of both waiting and riding time) of passengers into account. Since the more time the passenger spends in the system, the higher priorities the passenger can get, the request is more likely to be served first. The simulation results have shown this advantage of our method.

However, as the system considers both the location and time cost, which can be viewed as a compromise of both factors. And with the influx of new requests in the following slices, the unfinished requests from the time slices before cannot be guaranteed to be serviced firstly. Consequently, it is still possible for some of the passengers come earlier but could cost more time than some come later. Actually, there is existing method to further improve the performance of

our system. With the occasional use of extra vehicles, a time limit can be set to trigger the execution of the kind of backup vehicles. As a result, the service quality can be improved further more.

From the comparison of different routing algorithms, the insertion heuristic and the modified insertion heuristic routing algorithms are better when the volume of passengers is high, however, as the philosophy of this algorithm is based on pick-up first strategy (customer impatience), it has higher demand for the capacity of vehicles.

For the remaining two algorithms, within each isolated time slice, even though the composite pairing or-opt algorithm can get the solution much closer to the optimal than the modified pairing or-opt algorithm we proposed, the vehicles cannot finish all the routes designed by these algorithms in a single time slice in most cases. Consequently, from the standing point of the overall program, the near optimal solution is not the better choice for our solution framework, the algorithm we proposed is more suitable for our solution framework, which is the same for the insertion heuristic and modified insertion heuristic. The numerical results have reflected the same conclusion.

The computational complexity of our algorithms is  $O(n^3)$ , because the most complicated step for our algorithms is the assignment. As the complexity of Kuhn-Munkres algorithm is  $O(n^3)$ , our algorithms are also at the same level.

From the perspective of functions, our solution can provide services to all kinds of passengers in the city, not only the disabled. From the perspective of the budget, our solution can use either larger vehicles with high capacity or small vehicles with low capacity, it is a more flexible and economic solution.

Our solution has so many advantages, but it is still necessary to point out the disadvantages. In our system, the passengers whose requests are too far to finish in a vehicle in a single time slice may have the possibility to transfer to another vehicle, which will definitely cause the inconvenience and affect the user experience of our system. The second one is that we cannot give a guaranteed time limit to the passengers without using extra vehicles. It is because our solution framework is based on the order of priorities obtained from the Group Role Assignment, if the vehicle is forced to change the service order to reduce the time cost of a single request, it may cause the delay of more requests, which is not worthy for the whole system and is unfair for the remaining passengers. Consequently, those who are in rush are not suitable to use our system as their transit method.

Even though our solution has a great potential to provide service to the people who need to use the public transportation every day, it still cannot replace the traditional transit system. The equal pressure test in Chapter 6 has proven that with the high volume of passengers, the used time will be increased to an unacceptable level. As a result, our solution can be an auxiliary solution framework for the people who are not in a rush and want a better transit service, but cheaper than the taxi services.

Recently, our algorithm has provided an effective method to design a route for every vehicle in the system. However, with the help of parallel computing, the consumed time of this procedure will be reduced significantly.

Actually, the role range vector has not been used in our system right now, so the final assignment of our system in each time slice is a balanced assignment. One of the reasons is that we want to assign the jobs to different vehicles relatively balanced, so that the efficiency of all the



vehicles can be relatively balanced. Another reason is that we want to exploit the sharing ability of our system, as the vehicle routes are changing over time, the actual routes cannot be known beforehand, consequently, the best way is to assign as many passengers to vehicles as possible to exploit the limited transportation power and make more routes sharable. In the future, with the use of role range vector to regulate the number of passengers assigned to the vehicles, the fleet of vehicles can be heterogeneous, the system can have more flexibility to provide services to different types of people.

Moreover, with the help of vehicle positioning system, our method can use the instant location information as the input, and no longer need to restrict the speed of vehicles.

## References

- [1] Dantzig, George Bernard; Ramser, John Hubert (October 1959). The Truck Dispatching Problem. *Management Science*. 6 (1): 80–91.
- [2] V. Pillac et. al. A Review of Dynamic Vehicle Routing Problems. CIRRELT-2011-62.
- [3] D. J. Bertsimas and D. Simchi-Levi. A New Generation of Vehicle Routing Research: Robust Algorithms, Addressing Uncertainty. *Operations Research*. Vol. 44, No. 2, March-April 1996.
- [4] J. Bramel and D. Simchi-Levi. A Location Based Heuristic for General Routing Problems. *Operations Research* 43, 1995, 649-660.
- [5] M. L. Fisher and R. Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks* 11, 1981, 109-124.
- [6] H. Zhu, M. C. Zhou, and R. Alkins. Group role assignment via a Kuhn–Munkres algorithm-based solution. *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 42, no. 3, pp. 739–750, May 2012.
- [7] H. Zhu and M. C. Zhou. Role-based collaboration and its Kernel mechanisms. *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 36, no. 4, pp. 578–589, Jul. 2006.
- [8] Ross GT and Soland RM (1975). A Branch and Bound algorithm for the Generalized Assignment Problem. *Mathematical Programming* 8: 91-103.
- [9] Martello S and Toth P, 1981. An algorithm for the generalized Assignment Problem, In *Operations Research* 81, Brans JP (ed.), North-Holland, Amsterdam: 589-603.
- [10] Fisher ML, Jaikumar R and Van Wassenhove LN (1986). A Multiplier Adjustment Method for the Generalized Assignment Problem. *Management Science* 32(9): 1095-1103.
- [11] Guignard M and Rosenwein M (1989). An Improved Dual-Based Algorithm for the Generalized Assignment Problem. *Operations Research*, 37(4): 658-663.

- [12] NAUSS RM, 2003, Solving the generalized assignment problem: An optimizing heuristic approach. *INFORMS Journal of Computing* 15(3): 249-266.
- [13] Laguna M, Kelly JP, Gonzalez-Velarde JL & Glover FF (1995). Tabu search for the generalized assignment problem. *European Journal of Operational Research* 82: 176-189.
- [14] Savelsburgh M (1997). A branch and price algorithm for the generalized assignment problem. *Operations Research* 45(6): 831-841.
- [15] A. Ceselli and G. Righini. A Branch-and-Price Algorithm for the Multilevel Generalized Assignment Problem. *Operations Research* 54(6): 1172-1184.
- [16] J. A. Diaz and E. Fernandez. A Tabu search heuristic for the generalized assignment problem. *European Journal of Operational Research*, 132, pp. 22-38.
- [17] A. J. Higgins. A dynamic Tabu search for large-scale generalised assignment problems. *Computers and Operations Research*, 28, pp. 1039-1048.
- [18] Yagiura M, Ibaraki T and Glover F (2004). An ejection chain approach for the generalized assignment problem. *Inform Journal of Computing* 16: 133-151.
- [19] Chu PC & Beasley JE (1997). A genetic algorithm for the generalized assignment problem. *Computational Operations Research* 24: 17-23.
- [20] J. M. Wilson. A Genetic Algorithm for the generalised assignment problem. *Journal of the Operational Research Society*, 48, pp. 804-809.
- [21] G. R. Raidl and H. Feltl. An improved hybrid genetic algorithm for the generalized assignment problem. *Proceedings of the 2003 ACM Symposium on Applied Computing*, 2003-2004, ACM Press, pp. 990-995.
- [22] Osman IH (1995). Heuristics for the generalized assignment problem: Simulated annealing and tabu search approaches. *OR Spektrum* 17: 211-225.

- [23] Yagiura M, Ibaraki T and Glover F (2006). A path re-linking approach with ejection chains for the generalized assignment problem. *European Journal of Operational Research* 169: 548-569.
- [24] M. M. Amini and M. Racer. A hybrid heuristic for the generalized assignment problem. *European Journal of Operational Research*, 87, pp. 343-348.
- [25] Martello S and Toth P, 1981. An algorithm for the generalized Assignment Problem, In *Operations Research* 81, Brans JP (ed.), North-Holland, Amsterdam: 589-603.
- [26] YAGIURA M, Yamaguchi T and IBARAKI T (1998). A variable depth search algorithm with branching search for the generalized assignment problem. *Optimization Methods & Software* 10: 419-441.
- [27] M. A. Trick. A Linear Relaxation Heuristic for the Generalized Assignment Problem. *Naval Research Logistics*, 39, pp. 137-151.
- [28] H. R. Lourenco and D. Serra. Adaptive search heuristics for the generalized assignment problem. *Mathware and Soft Computing*, 9, pp. 209-234.
- [29] A. Baykasoglu et al. Artificial Bee Colony Algorithm and Its Application to Generalized Assignment Problem. *Swarm Intelligence: Focus on Ant and Particle Swarm Optimization*, pp. 113-144.
- [30] H. W. Kuhn. The Hungarian Method for the assignment problem. *Naval Research Logistics Quarterly*, 2: 83–97, 1955. Kuhn's original publication.
- [31] J. Munkres. Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957 March.
- [32] G. Laporte. The Vehicle Routing Problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* 59 (1992) 345-358.
- [33] G. Laporte and Y. Nobert. Exact algorithms for the vehicle routing problem. *Surveys in Combinatorial Optimization*, North-Holland, Amsterdam, 147-184.

- [34] N. Christofides and S. Eilon. An algorithm for the vehicle routing dispatching problem. *Operations Research Quaterly* 20, 309-318.
- [35] N. Christofides. The Vehicle Routing Problem. *RAIRO* 10, 55-70, 1976.
- [36] N. Christofides, A. Mingozzi and P. Toth. Exact Algorithms for the Vehicle Routing Problem Based on Spanning Tree Shortest Path Relaxations. *Mathematical Programming* 20, 255-282, 1981.
- [37] A. Kolen, A.H.G. Rinnooy Kan and H. Trienekens. Vehicle Routing with Time Windows, Report 843310, Emsmus University, Rotterdam, 1984.
- [38] N. Christofides. Vehicle Scheduling and Routing, presented at the 12th International Symposium of Mathematical Programming. Cambridge, Massachusetts, 1985.
- [39] H.N. Psaraftis. A Dynamic Programming Solution to the Single Vehicle Many-to-Many Immediate Request Dial-a-Ride Problem, *Transportation Science* 14, 130-154, 1980.
- [40] M. Held and R.M. Karp, A Dynamic Programming Approach to Sequencing Problems, *SIAM* 10, 196-210, 1962.
- [41] H.N. Psaraftis, ((An Exact Algorithm for the Single Vehicle Many-to-Many Dial-a-Ride Problem with Time Windows, *Transportation Science* 17, 351 - 360, 1983.
- [42] J. Desrosiers, Y. Dumas and F. Soumis. The Multiple Vehicles Many to Many Routing Problem with Time Windows, *Cahiersdu GERAD*, G-84-13, Ecole des Hautes Commerciales de Montrdal, 1984.
- [43] M. Balin'ski and R. Quandt. On an Integer Program for a Delivery Problem. *Operations Research*,12, 300-304, 1964.
- [44] M.R. Rao and S. Zionts. Allocation of Transportation Units to Alternative Trips - A Column Generation Scheme with Out-of-Kilter Subproblems, *Operations Research* 16, 52 - 63, 1968.

- [45] B.A. Foster and D.M. Ryan, An Integer Programming Approach to the Vehicle Scheduling Problem, *Operational Research Quarterly* 27, 367 - 384, 1976.
- [46] C. Orloff, Route Constrained Fleet Scheduling, *Transportation Science* 10, 149-168, 1976.
- [47] J. Desrosiers, F. Soumis and M. Desrochers, Routing with Time Windows by Column Generation, *Networks* 14, 545-565, 1984.
- [48] Y.K. Agarwal. Set Partitioning Approach to Vehicle Routing, presented at the TIMS/ORSA Conference, Boston, 1985.
- [49] M. L. Fisher and R. Jaikumar. A decomposition algorithm for large-scale vehicle routing. Working Paper 78-11-05, Department of Decision Sciences, University of Pennsylvania.
- [50] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik* 4, 238-252.
- [51] G. Laporte, Y. Nobert and M. Desrochers. Optimal routing under capacity and distance restrictions. *Operations Research* 33, 1050-1073.
- [52] W.M. Garvin, H.W. Crandall, J.B. John and R.A. Spellman. Applications of Linear Programming in the Oil Industry, *Management Science* 3, 407 - 430, 1957.
- [53] R. Baldacci, N. Chrisofides and A Mingozi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, Oct 2008, Vol 15, Issue 2, 351-385.
- [54] G. Laporte, M. Gendreau, J. Y. Potvin and F. Semet. Classical and Modern Heuristics for the Vehicle Routing Problem. *Les Cahiers du GERAD*, G-99-21.
- [55] Clarke, G., and Wright, J.W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research* 12, 568-581.
- [56] Gaskell, T. (1967). Bases for vehicle fleet scheduling. *Operational Research Quarterly* 18, 281-295.

- [57] Yellow, P. (1970). A computational modification to the savings method of vehicle scheduling. *Operational Research Quarterly* 21, 281-283.
- [58] Paessens, H. (1988). The savings algorithm for the vehicle routing problem. *European Journal of Operational Research* 34, 336-344.
- [59] Mole, R.H. and Jameson, S.R. (1976). A sequential route-building algorithm employing a generalized savings criterion, *Operational Research Quarterly* 27, 503–511.
- [60] Christofides, N. Mingozzi, A. and Toth, P. (1979). *The Vehicle Routing Problem. Combinatorial Optimization*. Wiley, Chichester.
- [61] Wren, A. (1971). *Computers in Transport Planning and Operation*. Ian Allan, London.
- [62] Wren, A. and Holliday, A. (1972). Computer scheduling of vehicles form one or more depots to a number of delivery points, *Operational Research Quarterly* 23, 333–344.
- [63] Gillett, B.E. and Miller, L.R. (1974). A heuristic algorithm for the vehicle dispatch problem, *Operations Research* 22, 240–349.
- [64] Bramel, J.B. and Simchi-Levi D. (1995). A location based heuristic for general routing problems, *Operations Research* 43, 649–660.
- [65] Lin, S. (1965). Computer solutions of the traveling salesman problem, *Bell System Technical Journal* 44, 2245–2269.
- [66] Gendreau, M. Hertz, A. and Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem, *Management Science* 40, 1276–1290.
- [67] Taillard, E.D. (1993). Parallel iterative search methods for vehicle routing problems, *Networks* 23, 661–673.
- [68] Xu, J. and Kelly, J.P. (1996). A network flow-based tabu search heuristic for the vehicle routing problem, *Transportation Science* 30, 379–393.

- [69] Rego, C. and Roucairol, C. (1996). A Parallel Tabu Search Algorithm using Ejection Chains for the Vehicle Routing Problem. In I.H. Osman and J.P. Kelly (eds), *Meta- Heuristics: Theory and Applications*. Kluwer, Boston.
- [70] Rochat, Y. and Taillard, 'E.D. (1995). Probabilistic diversification and intensification in local search for vehicle routing, *Journal of Heuristics* 1, 147–167.
- [71] Toth, P and Vigo, D (1998). Granular Tabu Search. Working paper, DEIS, University of Bologna.
- [72] N. Wilson and N. Colvin. Computer control of the Rochester dial-a-ride system. Technical Report R77-31.
- [73] H. Psaraftis. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2): 130-154.
- [74] V. Pillac et. al. A Review of Dynamic Vehicle Routing Problems. CIRRELT-2011-62.
- [75] P. V. Hentenryck and R. Bent. *Online stochastic combinatorial optimization*. MIT Press.
- [76] H. Psaraftis. Dynamic vehicle routing: status and prospects. *Annals of Operations Research*, 61(1): 143-164.
- [77] S. Ichoua, M. Gendreau and J. Y. Potvin. Planned route optimization for real-time vehicle routing. *Dynamic Fleet Management*, volume 38 of *Operations Research/Computer Science Interfaces*, 1-18. Springer US.
- [78] K. Lund, O. B. G. Madsen and J. M. Rygaard. Vehicle routing problems with varying degrees of dynamism. Technical report, Institute of Mathematical Modelling.
- [79] A. Larsen. The Dynamic Vehicle Routing Problem. PhD thesis, Technical University of Denmark.
- [80] J. Yang, P. Jaillet and H. Mahmassani. Real-time multivehicle truckload pickup and delivery problems. *Transportation Science*, 38(2): 135-148.



- [81] Z. Chen and H. Xu. Dynamic column generation for dynamic vehicle routing with time windows. *Transportation Science*, 40(1):74-88.
- [82] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2): 254-265.
- [83] M. Gendreau, F. Guertin, J. Y. Potvin and E. Taillard. Parallel tabu search for real-time vehicle routing and dispatching. *Transportation Science*, 33(4): 381-390.
- [84] R. Bent and P. V. Hentenryck. Scenario-based planning for partially dynamic vehicle routing with stochastic customers. *Operations Research*, 52(6): 977-987.
- [85] I. Benyahia and J. Y. Potvin. Decision support for vehicle dispatching using genetic programming. *IEEE Transactions on Systems Man and Cybernetics Part A-Systems and Humans*, 28(3): 306-314.
- [86] R. Montemanni, L. M. Gambardella, A. E. Rizzoli and A. V. Donati. Ant colony system for a dynamic vehicle routing problem. *Journal of Combinatorial Optimization*, 10(4): 327-343.
- [87] L. Gambardella et. al. Ant colony optimization for vehicle routing in advanced logistics systems. *Proceedings of the International Workshop on Modelling and Applied Simulation (MAS 2003)*: 3-9.
- [88] A. Rizzoli et. al. Ant colony optimization for real-world vehicle routing problems. *Swarm Intelligence*, 1:135-151.
- [89] T. Flatberg et. al. Dynamic and stochastic vehicle routing in practice. *Dynamic Fleet Management*, volume 38 of *Operations Research/Computer Science Interfaces*, 41-63. Springer US.
- [90] V. Pillac et. al. An event-driven optimization framework for dynamic vehicle routing. Technical report, France. Report 11/2/AUTO.
- [91] L. M. Hvattum et. al. Solving a dynamic and stochastic vehicle routing problem with a sample scenario hedging heuristic. *Transportation Science*, 40(4): 421-438.

- [92] G. Ghiani et. al. Anticipatory algorithms for same-day courier dispatching. *Transportation Research Part E: Logistics and Transportation Review*, 45(1): 96-106.
- [93] N. Azi et. al. A dynamic vehicle routing problem with multiple delivery routes. *Annals of Operations Research*, In-press.
- [94] S. Ichoua et. al. Exploiting knowledge about future demands for real-time vehicle dispatching. *Transportation Science*, 40(2): 211-225.
- [95] A. Attanasio et. al. Real-time fleet management at Ecourier Ltd. *Dynamic Fleet Management*, volume 38 of *Operations Research/Computer Science Interfaces*, chapter 10, 219-238. Springer US.
- [96] W. B. Powell et. al. Maximizing profits for North American Van Lines' truckload division: A new framework for pricing and operation. *Interfaces*, 18(1): 21-41.
- [97] B. W. Thomas and C. I. W. Chelsea. Anticipatory route selection. *Transportation Science*, 38(4): 473-487.
- [98] B. W. Thomas. Waiting strategies for anticipating service requests from known customer locations. *Transportation Science*, 41(3): 319-331.
- [99] C. Novoa and R. Storer. An approximate dynamic programming approach for the vehicle routing problem with stochastic demands. *European Journal of Operational Research*, 196(2): 509-515.
- [100] S. Yang et. al. Online dispatching and routing model for emergency vehicles with area coverage constraints. *Network Modeling 2005*, No. 1923 in *Transportation Research Record*, 1-8.
- [101] J. F. Cordeau and G. Laporte. The Dial-a-Ride Problem (DARP): Variants, modeling issues and algorithms. *4OR* 1:89-101 (2003).
- [102] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Bell Telephone Lab. (1979).

- [103] B. Kalantari, A. V. Hill and S. R. Arora. An algorithm for the traveling salesman problem with pickup and delivery customer. *European Journal of Operations Research*, Vol. 22 (1985). 377-386.
- [104] J. D. C. Little, K. G. Murty, D. M. Sweeney and C. Karel. An algorithm for the traveling salesman problem. *Operations Research*, Vol. 11 (1963). 972-989.
- [105] H. Suzuki and S. Nomura. A shortest path problem with visiting order constraints (In Japanese). *Proceedings of the 7<sup>th</sup> Mathematical Programming Symposium Japan* (1986). 127-142.
- [106] M. Kubo and H. Kasugai. Heuristic algorithms for the single vehicle dial-a-ride problem. *Journal of the Operations Research, Society of Japan*. Vol. 33, No. 4, December 1990.
- [107] H. Psaraftis. Analysis of an  $O(n^2)$  heuristic for the single vehicle many-to-many Euclidean Dial-a-ride Problem. *Transportation Science*. Vol. 117B (1983). 133-145.
- [108] H. Psaraftis. K-interchange procedures for local search in a precedence constrained routing problem. *European Journal of Operations Research*. Vol. 13 (1983). 391-402.
- [109] J. Jaw, A. R. Odoni, H. N. Psaraftis and N. H. M. Wilson. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research*. Vol. 20B (1986). 243-257.
- [110] D. Stein. Scheduling dial-a-ride transportation system. *Transportation Research*. Vol. 12 (1978). 232-249.
- [111] D. Stein. An asymptotic, probabilistic analysis of a routing problem. *Mathematics of Operations Research*. Vol. 3 (1978). 89-101.
- [112] R. Karp. Probabilistic analysis of partitioning algorithms for the traveling salesman in the plane. *Mathematics of Operations Research*. Vol. 2 (1977). 209-224.
- [113] C. F. Daganzo. An approximate analytic model of many-to-many demand responsive transportation systems. *Transportation Research*. Vol. 12 (1978). 325-333.

- [114] J. J. Bartholdi and L. K. Platzman. An  $O(n \log n)$  planar traveling salesman heuristic based on spacefilling curves. *Operations Research Letters*. Vol. 5 (1986). 165-169.
- [115] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys. The traveling salesman problem. *Operations Research*. Vol. 11 (1963). 972-989.
- [116] Bodin LD and Sexton T (1986). The multi-vehicle subscriber dial-a-ride problem. *TIMS Studies in Management Science* 2: 73–86.
- [117] Sexton T and Bodin LD (1985a). Optimizing single vehicle many-to-many operations with desired delivery times: I. Scheduling. *Transportation Science* 19: 378–410.
- [118] Sexton T and Bodin LD (1985b). Optimizing single vehicle many-to-many operations with desired delivery times: II. Routing. *Transportation Science* 19: 411–435.
- [119] Dumas Y, Desrosiers J, Soumis F (1989a). Large scale multi-vehicle dial-a-ride problems. *Les Cahiers du GERAD*, G–89–30, École des Hautes Études Commerciales, Montréal.
- [120] Dumas Y, Soumis F, Desrosiers J (1989b). Optimizing the schedule for a fixed vehicle path with convex inconvenience cost. *Les Cahiers du GERAD*, G–89–08, École des Hautes Études Commerciales, Montréal.
- [121] Desrosiers J, Dumas Y, Soumis F (1986). A dynamic programming solution of the large-scale single vehicle dial-a-ride problem with time windows. *American Journal of Mathematical and Management Sciences* 6: 301–325.
- [122] Desrosiers J, Dumas Y, Soumis F, Taillefer S, Villeneuve D (1991). An algorithm for mini-clustering in handicapped transport. *Les Cahiers du GERAD*, G–91-02, École des Hautes Études Commerciales, Montréal.
- [123] Ioachim I, Desrosiers J, Dumas Y, Solomon MM (1995). A request clustering algorithm for door-to-door handicapped transportation. *Transportation Science* 29: 63–78.

- [124] Toth P, Vigo D (1996). Fast local search algorithms for the handicapped persons transportation problem. In: Osman IH, Kelly JP (eds) *Meta-heuristics: Theory and applications*. Kluwer, Boston, 677– 690.
- [125] Toth P, Vigo D (1997). Heuristic algorithms for the handicapped persons transportation problem. *Transportation Science* 31: 60–71.
- [126] Borndörfer R, Klostermeier F, Grötschel M, Küttner C (1997). Telebus Berlin: Vehicle scheduling in a dial-a-ride system. Technical Report SC 97–23, Konrad-Zuse-Zentrum für Informationstechnik Berlin.
- [127] Wolfer Calvo R, Colomi A (2002). An approximation algorithm for the dial-a-ride problem. *Optimization Days 2002*, Montreal, Canada.
- [128] Cordeau J-F, Laporte G (2002). A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research B* (forthcoming).
- [129] Madsen OBG, Ravn HF, Rygaard JM (1995). A heuristic algorithm for the dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Annals of Operations Research* 60: 193–208.
- [130] Horn, M.E.T., 2002. Fleet scheduling and dispatching for demand-responsive passenger services. *Transportation Research Part C* 10, 35–63.
- [131] Beaudry, A., Laporte, G., Melo, T., Nickel, S., in press. Dynamic transportation of patients in hospitals. *OR Spectrum*.
- [132] Cordeau, J.-F., Laporte, G., Potvin, J.-Y., Savelsbergh, M.W.P., 2007. Transportation on demand. In: Barnhart, C., Laporte, G. (Eds.), *Transportation. Handbooks in Operations Research and Management Science*, vol. 14. North-Holland, Amsterdam. 429–466.
- [133] Attanasio, A., Cordeau, J.-F., Ghiani, G., Laporte, G., 2004. Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing* 30, 377–387.

[134] M. Caramia et. al. Routing a fleet of vehicles for dynamic combined pick-up and deliveries services. Operations Research Proceedings 2001, 3-8.

[135] Yi Cao. Hungarian Algorithm for Linear Assignment Problems (V2.3).

<https://www.mathworks.com/matlabcentral/fileexchange/20652-hungarian-algorithm-for-linear-assignment-problems-v2-3>.

## Appendix I

This appendix is the data set used in the simulation in Chapter 6, all the requests and the start vehicle location are generated randomly based on the grid of North Bay transit map. The time cost of all the requests are set to be 0 before they are put into the system.

Index of requests	Coordinates of pickup location		Coordinates of delivery location		Time cost (min)
	X1	Y1	X2	Y2	
1	11	13	8	4	0
2	12	8	16	11	0
3	15	12	8	5	0
4	12	24	16	13	0
5	9	9	17	18	0
6	11	24	9	7	0
7	6	3	13	22	0
8	1	24	18	12	0
9	10	5	6	3	0
10	1	24	10	13	0
11	6	3	11	5	0
12	11	5	13	6	0
13	12	8	17	12	0
14	13	22	1	24	0
15	17	18	3	5	0
16	1	24	15	21	0
17	11	11	7	14	0
18	11	24	13	18	0
19	1	24	3	5	0
20	11	8	19	13	0
21	13	13	15	12	0
22	1	21	14	17	0
23	11	19	18	12	0
24	11	8	15	21	0
25	8	1	15	12	0
26	8	15	10	13	0
27	12	24	14	17	0
28	9	11	1	21	0
29	16	13	11	13	0
30	18	12	11	8	0
31	11	24	8	4	0
32	12	19	1	21	0
33	15	10	10	20	0

34	9	7	16	13	0
35	6	19	12	5	0
36	12	8	15	10	0
37	3	5	10	20	0
38	11	19	8	15	0
39	13	6	10	13	0
40	12	8	13	22	0
41	8	15	15	12	0
42	12	19	11	1	0
43	13	15	6	3	0
44	16	11	8	5	0
45	11	1	1	21	0
46	13	6	14	16	0
47	12	13	19	13	0
48	9	11	8	11	0
49	13	22	15	10	0
50	13	6	11	8	0
51	17	12	12	19	0
52	12	5	15	10	0
53	12	8	14	16	0
54	12	24	16	11	0
55	19	13	11	5	0
56	16	13	10	13	0
57	13	15	13	25	0
58	10	5	11	19	0
59	15	21	9	20	0
60	8	5	17	18	0
61	1	21	13	6	0
62	9	7	12	19	0
63	12	5	15	21	0
64	16	17	6	17	0
65	11	1	15	21	0
66	9	20	1	21	0
67	11	5	11	11	0
68	9	9	7	14	0
69	12	19	12	24	0
70	10	13	4	19	0
71	10	13	13	22	0
72	16	17	3	5	0
73	12	8	19	13	0
74	8	1	11	19	0
75	12	13	15	12	0
76	11	1	8	15	0
77	14	16	17	12	0



78	17	18	7	14	0
79	1	24	13	22	0
80	11	1	14	16	0
81	13	25	4	19	0
82	15	12	13	22	0
83	13	18	6	17	0
84	10	13	9	9	0
85	8	5	9	20	0
86	19	13	15	10	0
87	6	3	13	15	0
88	13	25	14	11	0
89	6	19	9	11	0
90	11	5	12	24	0
91	13	6	8	4	0
92	8	4	16	13	0
93	15	21	8	1	0
94	11	5	12	5	0
95	1	21	9	9	0
96	3	5	9	7	0
97	15	12	13	6	0
98	13	25	8	5	0
99	9	11	19	13	0
100	1	24	9	11	0
101	13	6	11	5	0
102	11	1	1	21	0
103	17	12	11	19	0
104	12	5	14	16	0
105	13	22	9	7	0
106	8	4	9	20	0
107	12	19	17	18	0
108	9	11	19	13	0
109	13	13	9	20	0
110	12	19	1	21	0
111	13	18	16	13	0
112	8	4	17	18	0
113	6	3	13	18	0
114	12	8	8	15	0
115	11	11	11	13	0
116	4	19	14	11	0
117	15	21	11	1	0
118	7	14	14	17	0
119	14	16	6	19	0
120	12	8	8	5	0
121	1	21	7	14	0

122	10	5	16	17	0
123	12	5	18	12	0
124	16	13	10	13	0
125	17	12	16	13	0
126	17	18	10	5	0
127	11	11	7	14	0
128	12	24	12	8	0
129	11	8	8	4	0
130	17	18	16	11	0
131	19	13	14	16	0
132	15	21	11	5	0
133	15	10	16	13	0
134	16	11	9	7	0
135	6	17	14	16	0
136	9	11	12	5	0
137	17	12	6	17	0
138	19	13	12	19	0
139	17	12	19	13	0
140	15	12	8	5	0
141	11	24	11	13	0
142	1	21	9	9	0
143	16	17	17	18	0
144	15	10	14	11	0
145	9	11	14	16	0
146	6	19	19	13	0
147	12	19	9	9	0
148	3	5	16	11	0
149	15	10	14	16	0
150	11	24	8	15	0
151	8	4	11	13	0
152	10	20	12	19	0
153	6	19	1	21	0
154	9	11	8	5	0
155	16	17	12	13	0
156	1	24	12	5	0
157	8	5	12	5	0
158	8	5	8	11	0
159	10	13	9	20	0
160	12	19	15	21	0
161	11	19	13	18	0
162	12	19	11	19	0
163	14	11	15	21	0
164	15	12	17	12	0
165	8	11	15	21	0

166	14	16	17	12	0
167	12	19	11	8	0
168	16	13	8	1	0
169	11	11	14	11	0
170	14	16	8	11	0
171	15	12	18	12	0
172	10	20	11	5	0
173	10	20	17	18	0
174	1	24	16	11	0
175	8	1	16	13	0
176	9	11	15	12	0
177	17	12	16	13	0
178	10	13	9	20	0
179	12	24	14	16	0
180	4	19	12	8	0
181	14	17	15	12	0
182	1	24	13	25	0
183	18	12	14	17	0
184	17	18	12	5	0
185	7	14	11	24	0
186	6	17	13	18	0
187	11	24	1	24	0
188	12	8	13	13	0
189	14	16	11	8	0
190	3	5	18	12	0
191	12	24	3	5	0
192	14	11	12	24	0
193	13	6	1	21	0
194	13	13	12	13	0
195	14	16	10	13	0
196	18	12	8	1	0
197	11	11	8	15	0
198	12	24	13	22	0
199	13	6	8	11	0
200	10	13	8	15	0
201	11	11	14	11	0
202	12	5	10	20	0
203	12	5	1	24	0
204	12	8	12	13	0
205	11	5	15	21	0
206	14	17	9	7	0
207	13	15	1	24	0
208	11	8	13	18	0
209	13	15	6	3	0

210	8	4	10	13	0
211	12	24	12	5	0
212	9	11	14	17	0
213	14	16	8	5	0
214	15	12	12	8	0
215	11	19	10	13	0
216	8	1	17	12	0
217	9	7	13	25	0
218	9	9	11	19	0
219	1	21	8	1	0
220	17	18	4	19	0
221	12	5	6	3	0
222	8	4	11	5	0
223	11	24	1	24	0
224	6	19	11	8	0
225	11	1	17	12	0
226	11	5	1	24	0
227	11	1	19	13	0
228	11	11	12	24	0
229	3	5	16	11	0
230	9	11	3	5	0
231	10	13	19	13	0
232	10	20	12	19	0
233	13	15	13	18	0
234	9	9	8	1	0
235	1	21	12	5	0
236	13	15	13	6	0
237	15	21	11	24	0
238	13	25	7	14	0
239	7	14	16	13	0
240	6	3	17	12	0
241	11	24	12	5	0
242	9	11	10	5	0
243	6	17	11	24	0
244	15	12	3	5	0
245	13	15	17	18	0
246	15	12	14	17	0
247	11	19	13	22	0
248	14	17	9	11	0
249	10	13	12	13	0
250	8	5	17	18	0
251	16	13	11	1	0
252	15	10	1	24	0
253	6	19	11	8	0

254	11	19	13	25	0
255	16	17	1	21	0
256	8	1	1	24	0
257	13	6	11	5	0
258	7	14	6	3	0
259	8	11	12	24	0
260	11	13	12	19	0
261	8	11	9	11	0
262	10	5	16	17	0
263	6	3	11	11	0
264	15	12	8	4	0
265	16	17	17	18	0
266	13	18	16	13	0
267	16	13	14	16	0
268	8	11	8	5	0
269	10	20	18	12	0
270	9	9	15	12	0
271	11	11	13	6	0
272	9	11	12	5	0
273	9	9	11	24	0
274	16	13	14	17	0
275	16	17	10	13	0
276	13	13	6	3	0
277	1	21	19	13	0
278	13	15	13	25	0
279	11	1	9	7	0
280	18	12	11	8	0
281	14	17	12	13	0
282	13	6	8	4	0
283	8	15	14	11	0
284	15	21	12	24	0
285	14	16	1	21	0
286	7	14	11	13	0
287	8	1	6	3	0
288	13	13	8	15	0
289	6	3	13	22	0
290	9	20	13	15	0
291	12	19	17	12	0
292	15	12	17	18	0
293	10	5	8	11	0
294	6	19	8	5	0
295	10	20	13	18	0
296	16	13	8	5	0
297	9	9	17	18	0

298	3	5	17	12	0
299	19	13	1	24	0
300	11	5	11	24	0
301	11	8	11	11	0
302	16	13	13	13	0
303	10	20	17	12	0
304	13	25	14	17	0
305	9	20	13	18	0
306	8	15	13	13	0
307	8	4	13	6	0
308	17	18	8	15	0
309	11	24	14	11	0
310	10	13	1	24	0
311	15	10	12	5	0
312	8	4	9	7	0
313	12	8	1	24	0
314	6	19	7	14	0
315	9	9	6	19	0
316	10	20	12	19	0
317	6	3	10	20	0
318	14	11	13	25	0
319	12	24	14	17	0
320	12	8	9	11	0
321	14	11	11	11	0
322	7	14	1	21	0
323	15	12	12	24	0
324	9	20	15	12	0
325	15	12	7	14	0
326	4	19	9	20	0
327	11	11	16	11	0
328	6	3	12	13	0
329	11	19	8	5	0
330	1	21	13	15	0
331	12	8	8	1	0
332	1	24	1	21	0
333	11	13	9	7	0
334	8	1	11	1	0
335	17	18	9	9	0
336	11	13	9	11	0
337	11	24	14	17	0
338	10	20	1	21	0
339	17	12	8	4	0
340	16	17	14	16	0
341	12	8	18	12	0

342	8	11	17	12	0
343	1	21	10	13	0
344	10	5	14	11	0
345	1	24	4	19	0
346	12	5	7	14	0
347	16	11	14	16	0
348	19	13	11	8	0
349	11	8	1	21	0
350	6	17	16	11	0
351	13	6	14	11	0
352	11	24	9	20	0
353	13	22	13	18	0
354	11	24	7	14	0
355	6	17	9	9	0
356	7	14	11	24	0
357	10	13	11	1	0
358	12	5	1	21	0
359	18	12	13	6	0
360	11	8	7	14	0
361	12	5	1	21	0
362	6	19	14	16	0
363	9	9	14	17	0
364	11	1	9	7	0
365	9	7	1	21	0
366	11	8	12	8	0
367	13	22	9	7	0
368	13	15	14	17	0
369	13	18	4	19	0
370	1	24	13	22	0
371	17	18	13	6	0
372	12	8	12	13	0
373	13	13	13	15	0
374	6	3	11	11	0
375	6	19	8	1	0
376	16	11	8	5	0
377	19	13	12	13	0
378	9	20	11	5	0
379	15	12	11	5	0
380	9	9	12	8	0
381	1	21	11	1	0
382	3	5	12	19	0
383	9	7	1	24	0
384	14	11	4	19	0
385	11	13	13	13	0

386	8	11	10	13	0
387	16	11	8	15	0
388	12	24	6	3	0
389	10	5	14	16	0
390	13	13	6	17	0
391	17	12	9	20	0
392	4	19	10	5	0
393	13	18	13	15	0
394	6	17	9	7	0
395	13	22	13	18	0
396	4	19	1	24	0
397	16	13	8	5	0
398	15	12	8	4	0
399	11	11	12	19	0
400	11	11	12	19	0

The requests of 400 passengers.

Index of vehicles	Start coordinates of vehicles	
	X	Y
1	10	20
2	16	17
3	13	18
4	11	11
5	14	11
6	15	12
7	17	12
8	15	10
9	13	6
10	12	5
11	10	5
12	3	5
13	8	1

The start coordinates of 13 vehicles.



## Appendix II

This appendix is the simulation results of the differential pressure test using routing algorithm III in Chapter 6. All the 150 requests are coming from the first 150 of 400 requests in Appendix I. The start locations of all the 15 vehicles are also listed.

Index of requests	Coordinates of pickup location		Coordinates of delivery location		Time cost (min)
	X1	Y1	X2	Y2	
1	11	13	8	4	13.5
2	12	8	16	11	4.5
3	15	12	8	5	14.5
4	12	24	16	13	59
5	9	9	17	18	12
6	11	24	9	7	38
7	6	3	13	22	26
8	1	24	18	12	26
9	10	5	6	3	10.5
10	1	24	10	13	16
11	6	3	11	5	8.5
12	11	5	13	6	2
13	12	8	17	12	9
14	13	22	1	24	7
15	17	18	3	5	20
16	1	24	15	21	10
17	11	11	7	14	8
18	11	24	13	18	19.5
19	1	24	3	5	28.5
20	11	8	19	13	46.5
21	13	13	15	12	8.5
22	1	21	14	17	12.5
23	11	19	18	12	22
24	11	8	15	21	10
25	8	1	15	12	27.5
26	8	15	10	13	25
27	12	24	14	17	20.5
28	9	11	1	21	62.5
29	16	13	11	13	9
30	18	12	11	8	13
31	11	24	8	4	51.5
32	12	19	1	21	7
33	15	10	10	20	24.5

34	9	7	16	13	15
35	6	19	12	5	32
36	12	8	15	10	3.5
37	3	5	10	20	21.5
38	11	19	8	15	13.5
39	13	6	10	13	7
40	12	8	13	22	40
41	8	15	15	12	28
42	12	19	11	1	45
43	13	15	6	3	15
44	16	11	8	5	10
45	11	1	1	21	42
46	13	6	14	16	12.5
47	12	13	19	13	6
48	9	11	8	11	6
49	13	22	15	10	52.5
50	13	6	11	8	4
51	17	12	12	19	14
52	12	5	15	10	11.5
53	12	8	14	16	14.5
54	12	24	16	11	32
55	19	13	11	5	39.5
56	16	13	10	13	10
57	13	15	13	25	17
58	10	5	11	19	31
59	15	21	9	20	12.5
60	8	5	17	18	28
61	1	21	13	6	38
62	9	7	12	19	30
63	12	5	15	21	23.5
64	16	17	6	17	29
65	11	1	15	21	20
66	9	20	1	21	19.5
67	11	5	11	11	6.5
68	9	9	7	14	46
69	12	19	12	24	22.5
70	10	13	4	19	51.5
71	10	13	13	22	37
72	16	17	3	5	67
73	12	8	19	13	10.5
74	8	1	11	19	38.5
75	12	13	15	12	8
76	11	1	8	15	10
77	14	16	17	12	6.5

78	17	18	7	14	19.5
79	1	24	13	22	40
80	11	1	14	16	35.5
81	13	25	4	19	15.5
82	15	12	13	22	37
83	13	18	6	17	24
84	10	13	9	9	8.5
85	8	5	9	20	29.5
86	19	13	15	10	6
87	6	3	13	15	30
88	13	25	14	11	8.5
89	6	19	9	11	41
90	11	5	12	24	21.5
91	13	6	8	4	9.5
92	8	4	16	13	28
93	15	21	8	1	51.5
94	11	5	12	5	3
95	1	21	9	9	19.5
96	3	5	9	7	7.5
97	15	12	13	6	47
98	13	25	8	5	25.5
99	9	11	19	13	27.5
100	1	24	9	11	13.5
101	13	6	11	5	7.5
102	11	1	1	21	56.5
103	17	12	11	19	26
104	12	5	14	16	29.5
105	13	22	9	7	27
106	8	4	9	20	47
107	12	19	17	18	12
108	9	11	19	13	16.5
109	13	13	9	20	13
110	12	19	1	21	32.5
111	13	18	16	13	41.5
112	8	4	17	18	34
113	6	3	13	18	16
114	12	8	8	15	53.5
115	11	11	11	13	23
116	4	19	14	11	9.5
117	15	21	11	1	42
118	7	14	14	17	18.5
119	14	16	6	19	15
120	12	8	8	5	39
121	1	21	7	14	28.5

122	10	5	16	17	20
123	12	5	18	12	19.5
124	16	13	10	13	7.5
125	17	12	16	13	10
126	17	18	10	5	28
127	11	11	7	14	15
128	12	24	12	8	23
129	11	8	8	4	10
130	17	18	16	11	30
131	19	13	14	16	5.5
132	15	21	11	5	28.5
133	15	10	16	13	3
134	16	11	9	7	27.5
135	6	17	14	16	34.5
136	9	11	12	5	8
137	17	12	6	17	18
138	19	13	12	19	8
139	17	12	19	13	4.5
140	15	12	8	5	34
141	11	24	11	13	7.5
142	1	21	9	9	18.5
143	16	17	17	18	8.5
144	15	10	14	11	3
145	9	11	14	16	13
146	6	19	19	13	21.5
147	12	19	9	9	10.5
148	3	5	16	11	32
149	15	10	14	16	8.5
150	11	24	8	15	10

The requests and time cost of 150 passengers using routing algorithm III.

Index of requests	Coordinates of pickup location		Coordinates of delivery location		Time cost (min)
	X1	Y1	X2	Y2	
1	11	13	8	4	10.5
2	12	8	16	11	4.5
3	15	12	8	5	14.5
4	12	24	16	13	20
5	9	9	17	18	12
6	11	24	9	7	74.5
7	6	3	13	22	20
8	1	24	18	12	35.5
9	10	5	6	3	8.5
10	1	24	10	13	28
11	6	3	11	5	8.5
12	11	5	13	6	2
13	12	8	17	12	9
14	13	22	1	24	7
15	17	18	3	5	20
16	1	24	15	21	10
17	11	11	7	14	5
18	11	24	13	18	27.5
19	1	24	3	5	22.5
20	11	8	19	13	21.5
21	13	13	15	12	8.5
22	1	21	14	17	12.5
23	11	19	18	12	7
24	11	8	15	21	10
25	8	1	15	12	23.5
26	8	15	10	13	16
27	12	24	14	17	18.5
28	9	11	1	21	75
29	16	13	11	13	9
30	18	12	11	8	13
31	11	24	8	4	60.5
32	12	19	1	21	62.5
33	15	10	10	20	16.5
34	9	7	16	13	9
35	6	19	12	5	24
36	12	8	15	10	3.5
37	3	5	10	20	28.5
38	11	19	8	15	51
39	13	6	10	13	7
40	12	8	13	22	45
41	8	15	15	12	19

42	12	19	11	1	42
43	13	15	6	3	22
44	16	11	8	5	10
45	11	1	1	21	43
46	13	6	14	16	12.5
47	12	13	19	13	6
48	9	11	8	11	6
49	13	22	15	10	23
50	13	6	11	8	4
51	17	12	12	19	27
52	12	5	15	10	11
53	12	8	14	16	28.5
54	12	24	16	11	19
55	19	13	11	5	38.5
56	16	13	10	13	12
57	13	15	13	25	57.5
58	10	5	11	19	50
59	15	21	9	20	12.5
60	8	5	17	18	40
61	1	21	13	6	24.5
62	9	7	12	19	17
63	12	5	15	21	21
64	16	17	6	17	34
65	11	1	15	21	22.5
66	9	20	1	21	9
67	11	5	11	11	11
68	9	9	7	14	32
69	12	19	12	24	5.5
70	10	13	4	19	25
71	10	13	13	22	52.5
72	16	17	3	5	56.5
73	12	8	19	13	14.5
74	8	1	11	19	31.5
75	12	13	15	12	10
76	11	1	8	15	14
77	14	16	17	12	9
78	17	18	7	14	19.5
79	1	24	13	22	16.5
80	11	1	14	16	26.5
81	13	25	4	19	23.5
82	15	12	13	22	30
83	13	18	6	17	12
84	10	13	9	9	2.5
85	8	5	9	20	15.5

86	19	13	15	10	8
87	6	3	13	15	38.5
88	13	25	14	11	18
89	6	19	9	11	29
90	11	5	12	24	35.5
91	13	6	8	4	15.5
92	8	4	16	13	31.5
93	15	21	8	1	26.5
94	11	5	12	5	7
95	1	21	9	9	20.5
96	3	5	9	7	19.5
97	15	12	13	6	31.5
98	13	25	8	5	32.5
99	9	11	19	13	31.5
100	1	24	9	11	21.5
101	13	6	11	5	11.5
102	11	1	1	21	23.5
103	17	12	11	19	10.5
104	12	5	14	16	29.5
105	13	22	9	7	12
106	8	4	9	20	43
107	12	19	17	18	4
108	9	11	19	13	42.5
109	13	13	9	20	7
110	12	19	1	21	41
111	13	18	16	13	7
112	8	4	17	18	28.5
113	6	3	13	18	27
114	12	8	8	15	26
115	11	11	11	13	11.5
116	4	19	14	11	24
117	15	21	11	1	47
118	7	14	14	17	8
119	14	16	6	19	29
120	12	8	8	5	10
121	1	21	7	14	21.5
122	10	5	16	17	12.5
123	12	5	18	12	9
124	16	13	10	13	8
125	17	12	16	13	5
126	17	18	10	5	31
127	11	11	7	14	15
128	12	24	12	8	13.5
129	11	8	8	4	4.5

130	17	18	16	11	20
131	19	13	14	16	12.5
132	15	21	11	5	16
133	15	10	16	13	7
134	16	11	9	7	8.5
135	6	17	14	16	23.5
136	9	11	12	5	38
137	17	12	6	17	25
138	19	13	12	19	15
139	17	12	19	13	14
140	15	12	8	5	10
141	11	24	11	13	9.5
142	1	21	9	9	10
143	16	17	17	18	1
144	15	10	14	11	5
145	9	11	14	16	15.5
146	6	19	19	13	10.5
147	12	19	9	9	17.5
148	3	5	16	11	12
149	15	10	14	16	7.5
150	11	24	8	15	12

The requests and time cost of 150 passengers using routing algorithm IV.

Index of vehicles	Start coordinates of vehicles	
	X	Y
1	1	21
2	9	20
3	11	19
4	13	22
5	13	25
6	15	21
7	14	16
8	10	13
9	8	11
10	16	11
11	15	10
12	12	8
13	12	5
14	8	4
15	6	3

The start coordinates of 15 vehicles.



### Appendix III

This appendix is an example of 10 requests and uses 2 vehicles to serve. These requests are put into system in two continuous time slices with 5 requests each time. The speed of vehicles is set to be 60 km/h. The unit length of the grid is 0.5 km. Consequently, within each time slice (15 minutes), the vehicles can only travel 30 units in the grid. The vehicles need to spend 0.5 min travelling one unit length of the grid. This example uses the length of grid from both directional (y) and horizontal (x).

Index of requests	Coordinates of pickup location		Coordinates of delivery location		Time cost (min)
	X1	Y1	X2	Y2	
1	11	13	8	4	0
2	12	8	16	11	0
3	15	12	8	5	0
4	12	24	16	13	0
5	9	9	17	18	0
6	11	24	9	7	0
7	6	3	13	22	0
8	1	24	18	12	0
9	10	5	6	3	0
10	1	24	10	13	0

The requests of 10 passengers.

Index of vehicles	Start coordinates of vehicles	
	X	Y
1	1	21
2	9	20

The start coordinates of 2 vehicles.

In this example, the requests of index 1-5 are the input for the first time slice (15 min), the remaining are the input for the second time slice.

This is the first iteration.

The first step is to assign the requests to different vehicles. Preprocess the data as below:

Index of requests	d1		d2	t
	Vehicle 1	Vehicle 2		
1	18	9	12	0
2	24	15	7	0
3	23	14	14	0
4	14	7	15	0
5	20	11	17	0

1.  $d_1$  is the distance between the start location and the locations of all the buses;
2.  $d_2$  is the distance every passenger need to travel (from one's origin to one's destination);
3.  $t$  is the time cost of every passenger.

For the first two attributes, using the following equation to pre-process the data:

$$x_1 \text{ or } x_2 = \frac{d_{max} - d}{d_{max}} \quad (5.1.1)$$

Where  $d_{max}$  is the maximum distance in the graph, in our experiment it is set to 45.  $d$  is  $d_1$  or  $d_2$  with respect to  $x_1$  or  $x_2$ .

For the third attribute, using the following equation to get  $x_3$ :

$$x_3 = \frac{t}{t_{tor}} \quad (5.1.2)$$

where  $t_{tor}$  is the maximum tolerance time for all the passengers, in our experiment it is set to 150 min.

After preprocessing, the result is in the below table:

Index of requests	X1		X2	X3
	Vehicle 1	Vehicle 2		
1	(45-18)/45=0.6	(45-9)/45=0.8	(45-12)/45=0.73	0/150=0
2	(45-24)/45=0.47	(45-15)/45=0.67	(45-7)/45=0.84	0/150=0
3	(45-23)/45=0.49	(45-14)/45=0.69	(45-14)/45=0.69	0/150=0
4	(45-14)/45=0.69	(45-7)/45=0.84	(45-15)/45=0.67	0/150=0
5	(45-20)/45=0.56	(45-11)/45=0.76	(45-17)/45=0.62	0/150=0

After the pre-processing, the input data for GRA can be achieved using the equation:

$$input = c_1x_1 + c_2x_2 + c_3x_3 \quad (5.1.3)$$

With [0.4, 0.3, 0.3] as the coefficients, the result is as below:

Index of requests	Vehicle 1	Vehicle 2
1	0.460	0.540
2	0.440	0.520
3	0.402	0.482
4	0.476	0.538
5	0.409	0.489

Two vehicles to serve 5 passengers, it is definitely no longer a one-to-one assignment problem. With the help of GRAP, this assignment can be easily solved by duplicating the columns:

Index of requests	Vehicle 1	Vehicle 2	Vehicle 1	Vehicle 2	Vehicle 1
1	0.460	0.540	0.460	0.540	0.460
2	0.440	0.520	0.440	0.520	0.440
3	0.402	0.482	0.402	0.482	0.402
4	0.476	0.538	0.476	0.538	0.476
5	0.409	0.489	0.409	0.489	0.409

Then using Kuhn-Munkres algorithm, we can get the assignment:

Index of requests	Vehicle 1	Vehicle 2	Vehicle 1	Vehicle 2	Vehicle 1
1	0	1	0	0	0
2	0	0	0	1	0
3	0	0	0	0	1
4	0	0	1	0	0
5	1	0	0	0	0

Merge the result to get the final assignment for two vehicles:

Index of requests	Vehicle 1	Vehicle 2
1	0	1
2	0	1
3	1	0
4	1	0
5	1	0

As a result, requests 3, 4 and 5 are assigned to Vehicle 1, requests 1 and 2 are assigned to Vehicle 2.

The next step is to design routes for both vehicles.

For Vehicle 1, sort the info of requests based on the input of GRAP in a descending order:

Index of requests	Coordinates of pickup location		Coordinates of delivery location		Time cost (min)
	X1	Y1	X2	Y2	
4	12	24	16	13	0
5	9	9	17	18	0
3	15	12	8	5	0

Design route using Modified Pairing Or-opt algorithm:

Index of requests	Coordinates of routes	
	X1	Y1
4	12	24
4	16	13
3	15	12
3	8	5
5	9	9
5	17	18

The route Vehicle 1 can travel in this time slice (30 units):

Index of requests	Coordinates of routes	
	X1	Y1
0	1	21
4	12	24
4	16	13
0	15	13

Index 0 represents the current vehicle.

From the result we can see that in this time slice, passenger 4 has arrived, its information is deleted from pending matrix and get its time cost.

Index of requests	Time cost (min)
4	14.5

The next step is to update the information of passenger 3 and 5, the location of Vehicle 1 updates to (15, 13).

Index of requests	Coordinates of pickup location		Coordinates of delivery location		Time cost (min)
	X1	Y1	X2	Y2	
3	15	12	8	5	15
5	9	9	17	18	15

For Vehicle 2, the same procedure should be executed, some details entailed.

Design route using Modified Pairing Or-opt algorithm:

Index of requests	Coordinates of routes	
	X1	Y1
1	11	13
1	8	4
2	12	8
2	16	11

The route Vehicle 2 can travel in this time slice (30 units):

Index of requests	Coordinates of routes	
	X1	Y1
0	9	20
1	11	13
1	8	4
2	12	8
0	13	8

Passenger 2 has arrived its destination, its information is deleted from pending matrix and get its time cost.

Index of requests	Time cost (min)
1	10.5

Update Vehicle 2 location to (13, 8) and info of passenger 2.

Index of requests	Coordinates of pickup location		Coordinates of delivery location		Time cost (min)
	X1	Y1	X2	Y2	
2	13	8	16	11	15

This is the end of the first iteration.

This is the start of the second iteration.

The next step is to merge the info of not arrived passengers and the new requests.

Index of requests	Coordinates of pickup location		Coordinates of delivery location		Time cost (min)
	X1	Y1	X2	Y2	
2	13	8	16	11	15
3	15	12	8	5	15
5	9	9	17	18	15
6	11	24	9	7	0
7	6	3	13	22	0
8	1	24	18	12	0
9	10	5	6	3	0
10	1	24	10	13	0

The input data for GRA is:

Index of requests	Vehicle 1	Vehicle 2
2	0.628	0.690
3	0.628	0.583
5	0.528	0.572
6	0.440	0.413
7	0.358	0.420
8	0.284	0.258
9	0.544	0.607
10	0.344	0.318

We can get the assignment:

Index of requests	Vehicle 1	Vehicle 2
2	0	1
3	1	0
5	0	1
6	1	0
7	0	1
8	1	0
9	0	1
10	1	0

Split the info of passengers to different vehicles.

For Vehicle 1, passenger 3, 6, 8 and 10 are assigned.

Index of requests	Coordinates of pickup location		Coordinates of delivery location		Time cost (min)
	X1	Y1	X2	Y2	
3	15	12	8	5	15
6	11	24	9	7	0
10	1	24	10	13	0
8	1	24	18	12	0

Design route using Modified Pairing Or-opt algorithm:

Index of requests	Coordinates of routes	
	X1	Y1
3	15	12
6	11	24
10	1	24
8	1	24
8	18	12
10	10	13
6	9	7
3	8	5

The route Vehicle 1 can travel in this time slice (30 units):

Index of requests	Coordinates of routes	
	X1	Y1
0	15	13
3	15	12
6	11	24
10	1	24
8	1	24
0	4	24

No passengers have arrived. Update Vehicle 1 location to (4, 24) and info of passenger 3, 6, 8, 10.

Index of requests	Coordinates of pickup location		Coordinates of delivery location		Time cost (min)
	X1	Y1	X2	Y2	
3	4	24	8	5	30
6	4	24	9	7	15
8	4	24	18	12	15
10	4	24	10	13	15



For Vehicle 2, passenger 2, 5, 7 and 9 are assigned.

Index of requests	Coordinates of pickup location		Coordinates of delivery location		Time cost (min)
	X1	Y1	X2	Y2	
2	13	8	16	11	15
9	10	5	6	3	0
5	9	9	17	18	15
7	6	3	13	22	0

Design route using Modified Pairing Or-opt algorithm:

Index of requests	Coordinates of routes	
	X1	Y1
2	13	8
5	9	9
9	10	5
9	6	3
7	6	3
2	16	11
5	17	18
7	13	22

The route Vehicle 2 can travel in this time slice (30 units):

Index of requests	Coordinates of routes	
	X1	Y1
0	13	8
2	13	8
5	9	9
9	10	5
9	6	3
7	6	3
0	16	7

Passenger 9 has arrived its destination, its information is deleted from pending matrix and get their time cost.

Index of requests	Time cost (min)
9	8

Update Vehicle 2 location to (16, 7) and info of passenger 2, 5, 7.

Index of requests	Coordinates of pickup location		Coordinates of delivery location		Time cost (min)
	X1	Y1	X2	Y2	
2	16	7	16	11	30
5	16	7	17	18	30
7	16	7	13	22	15

This is the end of the second iteration.

This is the start of the third iteration.

The next step is to merge the info of not arrived passengers.

Index of requests	Coordinates of pickup location		Coordinates of delivery location		Time cost (min)
	X1	Y1	X2	Y2	
2	16	7	16	11	30
3	4	24	8	5	30
5	16	7	17	18	30
6	4	24	9	7	15
7	16	7	13	22	15
8	4	24	18	12	15
10	4	24	10	13	15

The input data for GRA is:

Index of requests	Vehicle 1	Vehicle 2
2	0.476	0.733
3	0.607	0.349
5	0.422	0.680
6	0.583	0.326
7	0.352	0.610
8	0.557	0.299
10	0.617	0.359

we can get the assignment:

Index of requests	Vehicle 1	Vehicle 2
2	0	1
3	1	0
5	0	1

6	1	0
7	0	1
8	1	0
10	1	0

Split the info of passengers to different vehicles.

For Vehicle 1, passenger 3, 6, 8 and 10 are assigned.

Index of requests	Coordinates of pickup location		Coordinates of delivery location		Time cost (min)
	X1	Y1	X2	Y2	
10	4	24	10	13	15
3	4	24	8	5	30
6	4	24	9	7	15
8	4	24	18	12	15

Design route using Modified Pairing Or-opt algorithm:

Index of requests	Coordinates of routes	
	X1	Y1
10	4	24
3	4	24
6	4	24
8	4	24
3	8	5
6	9	7
10	10	13
8	18	12

The route Vehicle 1 can travel in this time slice (30 units):

Index of requests	Coordinates of routes	
	X1	Y1
0	4	24
10	4	24
3	4	24
6	4	24
8	4	24
3	8	5
6	9	7
0	10	12

Passenger 3 and 6 have arrived their destinations, their information is deleted from pending matrix and get their time cost.

Index of requests	Time cost (min)
3	41.5
6	28

Update Vehicle 1 location to (10, 12) and info of passenger 8 and 10.

Index of requests	Coordinates of pickup location		Coordinates of delivery location		Time cost (min)
	X1	Y1	X2	Y2	
8	10	12	18	12	30
10	10	12	10	13	30

For Vehicle 2, passenger 2, 5 and 7 are assigned.

Index of requests	Coordinates of pickup location		Coordinates of delivery location		Time cost (min)
	X1	Y1	X2	Y2	
2	16	7	16	11	30
5	16	7	17	18	30
7	16	7	13	22	15

Design route using Modified Pairing Or-opt algorithm:

Index of requests	Coordinates of routes	
	X1	Y1
2	16	7
5	16	7
7	16	7
2	16	11
5	17	18
7	13	22

The route Vehicle 2 can travel in this time slice (30 units):

Index of requests	Coordinates of routes	
	X1	Y1
0	16	7
2	16	7
5	16	7
7	16	7
2	16	11
5	17	18
7	13	22
0	13	22

All the passengers have arrived, their information is deleted from pending matrix and get their time cost.

Index of requests	Time cost (min)
2	32
5	36
7	25

Update Vehicle 2 location to (13, 22).

This is the end of third iteration.

This is the start of fourth iteration.

The next step is to merge the info of not arrived passengers.

Index of requests	Coordinates of pickup location		Coordinates of delivery location		Time cost (min)
	X1	Y1	X2	Y2	
8	10	12	18	12	30
10	10	12	10	13	30

The input data for GRA is:

Index of requests	Vehicle 1	Vehicle 2
8	0.707	0.591
10	0.753	0.638

we can get the assignment:

Index of requests	Vehicle 1	Vehicle 2
8	0	1
10	1	0

Split the info of passengers to different vehicles.

For Vehicle 1, passenger 10 is assigned.

Index of requests	Coordinates of pickup location		Coordinates of delivery location		Time cost (min)
	X1	Y1	X2	Y2	
10	10	12	10	13	30

Design route using Modified Pairing Or-opt algorithm:

Index of requests	Coordinates of routes	
	X1	Y1
10	10	12
10	10	13

The route Vehicle 1 can travel in this time slice (30 units):

Index of requests	Coordinates of routes	
	X1	Y1
0	10	12
10	10	12
10	10	13
0	10	13

Passenger 10 has arrived, the information is deleted from pending matrix and get the time cost.

Index of requests	Time cost (min)
10	30.5

Update Vehicle 1 location to (10, 13).

For Vehicle 2, passenger 8 is assigned.

Index of requests	Coordinates of pickup location		Coordinates of delivery location		Time cost (min)
	X1	Y1	X2	Y2	
8	10	12	18	12	30

Design route using Modified Pairing Or-opt algorithm:

Index of requests	Coordinates of routes	
	X1	Y1
8	10	12
8	18	12

The route Vehicle 2 can travel in this time slice (30 units):

Index of requests	Coordinates of routes	
	X1	Y1
0	13	22
8	10	12
8	18	12
0	18	12

Passenger 8 has arrived. The information is deleted from pending matrix and get the time cost.

Index of requests	Time cost (min)
8	40.5

Update Vehicle 2 location to (18, 12).

This is the end of fourth iteration.

This is the end of the whole program.

The time cost of all the passengers is in the below table.

Index of requests	Time cost (min)
1	10.5
2	32
3	41.5
4	14.5
5	36
6	28
7	25
8	40.5
9	8
10	30.5

This is the end of Appendix III.



## Appendix IV

These are the main codes used in this thesis.

```
function [result, busLocation, rTraveled] = GRArouting(f, b)
% Group Role Assignment Routing Key Function
% This function is the main function of GRA routing project, every 15min loads k passengers, it will
stop until all the passengers arrive their destinations.
%
% Parameters:
%   input:
%       f ---- original info of passengers(n by 6)
%       b ---- matrix of all the bus locations(n by 2)
%   output:
%       result ---- final result of passengers(n by 6)
%       busLocation ---- final position of all the buses(n by 2)
%
% Example:
%
% NOTE:
%
% Author: Bo Lei
% Email: blei@laurentian.ca
% Website:
%
% Copyright (C) 2017-2018 Bo-Lei. All rights reserved.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% log:
% 2017-10-18: Complete
% 2018-02-27: Modified to 'cell' type
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

tic

result = [];
x = b; % a copy of b
nP = size(f, 1); % number of all the passengers
K = 100;
J = K;
kF = f(1:K, :); % load the first 'K' passengers
r = size(b, 1); % number of buses
rTraveled = cell(1, r);

while ~isempty(kF)

    [cost] = evaluate(kF, b);
    [finalAssignment, ] = assignbus(cost);
    H = userapportion1(kF, finalAssignment, cost, r);
```

```

c = size(H, 2); % number of buses apportioned
nArrived = [];
arrived = [];
h = cell(1, r);
t = cell(1, c);
g = t;
z = t;
L = t;

% design all the bus routes and update the info
for i = 1:c
    if ~isempty(H{i}) % make sure there is at least one passenger in bus
        h{i} = busroute3(b(i, :), H{i}(:, 1:5));
        [t{i}, g{i}, z{i}, L{i}] = infoupdate(b(i,:), h{i}, H{i});
        nArrived = [nArrived; t{i}]; % merge all the passengers not arrived in one matrix
        arrived = [arrived; g{i}]; % merge all the passengers arrived in one matrix
        b(i, :) = z{i}; % update bus location
        rTraveled{i} = [rTraveled{i}; L{i}]; % record every bus route has traveled

    else % there is no passenger in this bus
        i = i + 1;
    end
end

result = [result; arrived];

if K ~= nP
    if (K+J) < nP
        kF = [nArrived; f((K+1): (K+J), :)];
        K = K + J;
    else
        kF = [nArrived; f((K+1): nP, :)];
        K = nP;
    end
else
    kF = nArrived;
end
end

busLocation = b;

end

toc

```

```

function [cost] = evaluate(f, b)
% get the evaluate matrix of all the passengers for different buses
%
% Parameters:

```

```

% input:
% f ---- original info of passengers(n by 6)
% b ---- matrix of all the bus locations(n by 2)
% output:
% cost ---- cost matrix of all the input passengers
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

rb = size(b, 1);
rf = size(f, 1);
cost = zeros(rf, rb);

for i = 1 : rf
    for j = 1 : rb
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
        % cost(i, j) = 0.3 * distance from start location of every passenger to each bus
        % + 0.4 * distance from end location of every passenger to each bus
        % + 0.3 * time consuming
        cost(i, j) = (45 - (abs(f(i, 2) - b(j, 1)) + abs(f(i, 3) - b(j, 2))))/45*0.4
            + (45 - (abs(f(i, 4) - f(i, 2)) + abs(f(i, 5) - f(i, 3))))/45*0.3
            + f(i, 6)/150*0.3;
    end
end
end

```

```

function [finalAssignment, cost] = assignbus(costMat)
% assignbus Using Munkres Algorithm to assign passengers to buses
%
% [finalAssignment, cost] = assignbus(costMat) returns the optimal assignment in ASSIGN
% with the maximam cost based on the assignment problem represented by the
% costMat, where the (i,j)th element represents the cost to assign the i-th
% passenger to the j-th bus. All the elements in costMat are not larger than
% 1.

% This is vectorized implementation of the algorithm [135]. It is the fastest
% among all Matlab implementations of the algorithm.

%
% Parameters:
% input:
% costMat ---- cost matrix
% output:
% finalAssignment ---- final result to assign ith
% passenget to jth bus
% cost ---- total cost of all the passengers
% Example:
%
% NOTE:
%
% Author: Bo Lei

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% log:
% 2017-08-27: Complete
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[m,n] = size(costMat);

% copy of dimensions
a = m;
b = n;

% number of passengers not equal to buses
if m <= n
    costMat = [costMat; zeros(n-m,n)];

else % m>n, duplicate columns to obtain a square matrix
    for i = (n+1):m
        if (mod(i,n) == 0)
            costMat(:, i) = costMat(:, n);
        else
            costMat(:, i) = costMat(:, mod(i, n));
        end
    end
end

% to get the maximum, keep a copy of input matrix and obtain a reverse
% value of costMat.
costMat1 = costMat;
costMat = 1 - costMat;

% Check the input matrix not include infinity
assignment = false(size(costMat));
cost = 0;

costMat(costMat~=costMat)=Inf;
validMat = costMat<Inf;
validCol = any(validMat);
validRow = any(validMat,2);

nRows = sum(validRow);
nCols = sum(validCol);
n = max(nRows,nCols);
if ~n
    return
end

dMat = zeros(n);
dMat(1:nRows,1:nCols) = costMat(validRow,validCol);

```

```

% *****
% Munkres' Assignment Algorithm starts here
% *****

%%%%%%%%%%%%%%
% STEP 1: Subtract the row minimum from each row.
%%%%%%%%%%%%%%
dMat = bsxfun(@minus, dMat, min(dMat,[],2));

% *****
% STEP 2: Find a zero of dMat. If there are no starred zeros in its
%         column or row start the zero. Repeat for each zero
% *****
zP = ~dMat; % set all zeros 1, all nonzeros 0
starZ = false(n);
while any(zP(:))
    [r,c]=find(zP,1);
    starZ(r,c)=true;
    zP(r,:)=false;
    zP(:,c)=false;
end

while 1
    % *****
    % STEP 3: Cover each column with a starred zero. If all the columns are
    %         covered then the matching is maximum
    % *****
    primeZ = false(n);
    coverColumn = any(starZ);
    if ~any(~coverColumn)
        break
    end
    coverRow = false(n,1);
    while 1
        % *****
        % STEP 4: Find a noncovered zero and prime it. If there is no starred
        %         zero in the row containing this primed zero, Go to Step 5.
        %         Otherwise, cover this row and uncover the column containing
        %         the starred zero. Continue in this manner until there are no
        %         uncovered zeros left. Save the smallest uncovered value and
        %         Go to Step 6.
        % *****
        zP(:) = false;
        zP(~coverRow,~coverColumn) = ~dMat(~coverRow,~coverColumn);
        Step = 6;
        while any(any(zP(~coverRow,~coverColumn)))
            [uZr,uZc] = find(zP,1);
            primeZ(uZr,uZc) = true;

```

```

    stz = starZ(uZr,:);
    if ~any(stz)
        Step = 5;
        break;
    end
    coverRow(uZr) = true;
    coverColumn(stz) = false;
    zP(uZr,:) = false;
    zP(~coverRow,stz) = ~dMat(~coverRow,stz);
end
if Step == 6
    % *****
    % STEP 6: Add the minimum uncovered value to every element of each covered
    % row, and subtract it from every element of each uncovered column.
    % Return to Step 4 without altering any stars, primes, or covered lines.
    % *****
    M=dMat(~coverRow,~coverColumn);
    minval=min(min(M));
    if minval==inf
        return
    end
    dMat(coverRow,coverColumn)=dMat(coverRow,coverColumn)+minval;
    dMat(~coverRow,~coverColumn)=M-minval;
else
    break
end
end
% *****
% STEP 5:
% Construct a series of alternating primed and starred zeros as
% follows:
% Let Z0 represent the uncovered primed zero found in Step 4.
% Let Z1 denote the starred zero in the column of Z0 (if any).
% Let Z2 denote the primed zero in the row of Z1 (there will always
% be one). Continue until the series terminates at a primed zero
% that has no starred zero in its column. Unstar each starred
% zero of the series, star each primed zero of the series, erase
% all primes and uncover every line in the matrix. Return to Step 3.
% *****
rowZ1 = starZ(:,uZc);
starZ(uZr,uZc)=true;
while any(rowZ1)
    starZ(rowZ1,uZc)=false;
    uZc = primeZ(rowZ1,:);
    uZr = rowZ1;
    rowZ1 = starZ(:,uZc);
    starZ(uZr,uZc)=true;
end
end
end

```

```
% Cost of assignment
assignment(validRow,validCol) = starZ(1:nRows,1:nCols);
cost = sum(costMat1(assignment));
```

```
finalAssignment = zeros(a, b);
```

```
% number of passengers not equal to buses
```

```
if a <= b
```

```
    finalAssignment = assignment(1:a, 1:b);
```

```
else % m>n, restore the original dimensions
```

```
    finalAssignment(:, 1:b) = assignment(:, 1:b);
```

```
    for i = 1:m
```

```
        for j = b+1:m
```

```
            if (assignment(i, j) == 1 && mod(j, b) ~= 0)
```

```
                finalAssignment(i, mod(j, b)) = 1;
```

```
            elseif (assignment(i, j) == 1 && mod(j, b) == 0)
```

```
                finalAssignment(i, b) = 1;
```

```
            end
```

```
        end
```

```
    end
```

```
end
```

```
end
```

```
function H = userapportion1(f, g, costMat, n)
```

```
% apportion passengers into 'n' buses according to costMat and time-consuming in f, obtain a
```

```
% descending passengers' info matrix
```

```
%
```

```
% Parameters:
```

```
% input:
```

```
% f ---- input matrix(one matrix containing index, locations and
```

```
% time-consuming) (n by 6 matrix)
```

```
% g ---- input matrix(assignbus)
```

```
% costMat ---- cost matrix of every passenger
```

```
% n ---- number of buses
```

```
% output:
```

```
% H ---- passengers' info in 'n' buses(cell type)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
for i = 1 : n
```

```
    row_index{i} = [g(:, i) == 1];
```

```
    a{i} = [costMat(row_index{i}, 1), f(row_index{i}, :)]; % merge the cost and f into one matrix
```

```
    s{i} = sortrows(a{i}, [-1 -7]); % descend sort based on cost and time-consuming
```

```
    H{i} = s{i}(:, 2:7); % obtain info of passengers
```

```
end
```

```

function [h] = busroute3(a,b)
% design the bus route based on the third proposed algorithm in Chapter 5
%
% Parameters:
%   input:
%       a ---- input bus location(1 by 2 matrix)
%       b ---- input matrix containing passengers' info(n by 5 matrix)
%   output:
%       h ---- route matrix(2n by 3)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
s = busroute2(b); % route obtained from busroute2
bus = [0, a];
r = [bus; s];

% preallocate c as the route matrix
row = size(r, 1);
c = zeros(row, 6);
c(:, 1:3) = r;

c(1, 4) = 0;
c(1, 6) = 0;

for i = 2: row
    c(i, 4) = abs(c(i-1, 2) - c(i, 2)) + abs(c(i-1, 3) - c(i, 3));
    c(i, 6) = c(i, 4) + c(i-1, 6);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% determine the bus location after running 15 min
% from x to y
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% set bus speed as 60 km/h, in a 20*30 grid, length per unit is 500 meters,
% bus can travel 30 units every 15 min
c(1, 5) = 30;

for j = 2: row
    for k = 1: j
        c(j, 5) = c(j-1, 5) - c(k, 4);
    end
end

[x, ] = find(c(:, 5) < 0, 1); % find the first negative value in 5th column
p = unique(c(2:(x-1), 1), 'stable');
k = size(p, 1);
b1 = [];
for i = 1:k
    tempb = b(b(:, 1) == p(i), :);
    b1 = [b1; tempb];
end

```



```

end

if ~isempty(b1)
    b1 = sortrows(b1,1);
    b1 = b(ismember(b, b1, 'rows'),:);
    b2 = setdiff(b,b1,'rows');
    h1 = busroute2(b1);
    h2 = busroute2(b2);
    h = [h1;h2];
else
    h = busroute2(b);
end

end

```

```

function [h] = busroute2(b)
% design the bus route based on the second proposed algorithm in Chapter 5
%
% Parameters:
%   input:
%       b ---- input matrix containing passengers' info(n by 5 matrix)
%   output:
%       h ---- route matrix(2n by 3)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

r = size(b,1);
x = 1:r;
x = x';
b1 = [x,b];
r1 = 2*r;
initialR = zeros(r1, 4); %initial route containing index

if r == 1
    h = [b1(1,1:4); b1(1,[1 2 5 6])];
else
    for i = 1:r1
        if i <= r
            if mod(i,2) == 1
                initialR(i, :) = b1(floor((i+1)/2), 1:4);
            else
                initialR(i, :) = b1(floor(i/2), [1 2 5 6]);
            end
        elseif mod(i,2) == 1
            initialR(i, :) = b1(floor((i+1)/2), 1:4);
        else
            initialR(i, :) = b1(floor(i/2), [1 2 5 6]);
        end
    end
end
end

```

```

h = initialR;
%two route variants for TS
h1 = initialR;
h2 = initialR;
cinitialR = initialR;
optimalRL = routelength(initialR); %initial route length as optimal route length

if r >= 2
    c = nchoosek(1:r,2); %all the pairs of customers
    k = size(c, 1);
    for i = 1:k
        t1 = find(initialR(:,1) == c(i, 1),1,'last'); % row of p1 destination
        t2 = find(initialR(:,1) == c(i, 2),1,'first'); % row of p2 start
        t3 = find(initialR(:,1) == c(i, 2),1,'last'); % row of p2 destination
        if t1 < t2
            initialR([t1 t2],:) = initialR([t2 t1],:);
            h1 = initialR;
            s1 = routelength(h1);
            initialR([t2 t3],:) = initialR([t3 t2],:);
            h2 = initialR;
            s2 = routelength(h2);
            if s1 <= s2
                if s1 < optimalRL
                    optimalRL = s1;
                    h = h1;
                end
            elseif s2 < optimalRL
                optimalRL = s2;
                h = h2;
            end
            end
        initialR = h;
    end

    %%%%%%%%%%%%%%%
    % optimize the route of all the pick-ups before first drop-off
    %%%%%%%%%%%%%%%
    [ival, irow] = unique(initialR(:,2), 'first');
    rowindex = setdiff(1:numel(initialR(:,2)), irow);
    secrowindex = rowindex(1); % first repeated row index

    if secrowindex > 2
        combo = nchoosek(1:(secrowindex-1),2); %all the pairs of pickups before first dropoff
        k = size(combo, 1);

        for i = 1:k
            t1 = combo(i, 1); % row of p1 start
            t2 = combo(i, 2); % row of p2 start

```

```

        initialR([t1 t2],:) = initialR([t2 t1],:);
        h1 = initialR;
        s1 = routelength(h1);
        if s1 < optimalRL
            optimalRL = s1;
            initialR = h1;
        else
            initialR([t1 t2],:) = initialR([t2 t1],:);
        end
    end
    h = initialR;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% optimize the route of all the drop-offs after last pick-up
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[jval, jrow] = unique(initialR(:,2), 'first');
lastpickupindex = max(jrow); % row of last pick-up
tempmatrix = initialR((lastpickupindex+1):end, :);
numofrows = size(tempmatrix,1);
if numofrows>1
    c1 = nchoosek(1:numofrows,2);
    j = size(c1,1);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i = 1:j
        t1 = (c1(i, 1) + lastpickupindex); % row of p1 destination
        t2 = (c1(i, 2) + lastpickupindex); % row of p2 destination
        s = routelength(initialR);
        initialR([t1 t2],:) = initialR([t2 t1],:);
        h3 = initialR;
        s3 = routelength(h3);
        if s3 < s
            optimalRL = s3;
            h = h3;
            initialR = h3;
        else
            initialR([t1 t2],:) = initialR([t2 t1],:);
        end
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
else
    h = [b1(1,1:4); b1(1,[1 2 5 6])];
end
h = h(:,2:end);
end

```

```

function s = routelength(h)
% calculate the length of route
%
% Parameters:
%   input:
%       h ---- route matrix(n by 4) contain index
%   output:
%       s ---- route length
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
a = size(h, 1);
L = zeros(a, 1);
s = 0;
for i = 1:(a-1)
    L(i,1) = abs(h(i+1, 3) - h(i,3)) + abs(h(i+1, 4) - h(i,4));
    s = s + L(i,1);
end

end

```

```

function [t, g, z, L] = infoupdate(a, h, f)
% update info of passengers and bus
%
% Parameters:
%   input:
%       a ---- input bus location(1 by 2 matrix)
%       h ---- route matrix(2n by 3)
%       f ---- original info of passengers(n by 6)
%   output:
%       t ---- info of not arrived passengers(n by 6)
%       g ---- info of arrived passengers(n by 6)
%       z ---- output bus location(1 by 2 matrix)
%       L ---- route bus has traveled(n by 3 matrix)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% set index of original bus location as 0, insert it into route
b = [0, a];
r = [b; h];

% preallocate c as the route matrix
row = size(r, 1);
c = zeros(row, 6);
c(:, 1:3) = r;
c(1, 4) = 0;
c(1, 6) = 0;

for i = 2: row
    c(i, 4) = abs(c(i-1, 2) - c(i, 2)) + abs(c(i-1, 3) - c(i, 3));
    c(i, 6) = c(i, 4) + c(i-1, 6);
end

end

```

```

% set bus speed as 60 km/h, in a 20*30 grid, length per unit is 500 meters,
% bus can travel 30 units every 15 min
c(1, 5) = 30;

for j = 2: row
    for k = 1: j
        c(j, 5) = c(j-1, 5) - c(k, 4);
    end
end

[x, ] = find(c(:, 5) < 0, 1); % find the first negative value in 5th column

if ~isempty(x) % passengers can not to be serviced in one iteration(15 min)
    m = x - 1;
    if m ~= 1 % bus has traveled and pick up at least 1 passenger
        if c(m, 5) == 0
            z = c(m, 2:3);
        elseif abs(c(x, 2) - c(m, 2)) >= c(m, 5) && (c(x, 2) - c(m, 2)) >= 0
            z = [c(m, 2) + c(m, 5), c(m, 3)];
        elseif abs(c(x, 2) - c(m, 2)) >= c(m, 5) && (c(x, 2) - c(m, 2)) < 0
            z = [c(m, 2) - c(m, 5), c(m, 3)];
        elseif (c(x, 3) - c(m, 3)) >= 0
            z = [c(x, 2), c(m, 3) + c(m, 5) - abs(c(x, 2) - c(m, 2))];
        else
            z = [c(x, 2), c(m, 3) - abs(c(x, 2) - c(m, 2))];
        end

        % route bus has traveled, not including the final location,
        % because it is also the start location of next iteration
        L = c(1:m, 1:3);

        % count the frequency to decide the conditions of passengers
        table = tabulate(L(2:m, 1)/2);
        table(:, 1) = table(:, 1) * 2;
        table1 = sortrows(table, -1);
        y = size(table1, 1);
        g = [];
        t1 = [];

        %%%%%%%%%%%%%%%
        % update info of passengers
        for n = 1: y
            if table1(n, 2) == 2 % arrived passengers
                id = f(:, 1) == table1(n, 1);
                g = [g; f(id, :)]; % wrong
                f(id, :) = [];

            else % not arrived passengers
                id = f(:, 1) == table1(n, 1);
            end
        end
    end
end

```

```

        t1 = [t1; f(id, :)]; % wrong
        f(id, :) = [];
    end
end

% info of arrived passengers
if ~isempty(g)
    g(:, 2:3) = g(:, 4:5); % they have arrived, so their instant location is same as destination.
    [ig, ] = size(g);
    for i = 1: ig
        idg = find(c(:, 1) == g(i, 1), 1, 'last');
        g(i, 6) = g(i, 6) + 0.5 * c(idg, 6);
    end
end

if ~isempty(t1)
    % info of not arrived but has traveled passengers
    t1(:, 2) = z(1, 1);
    t1(:, 3) = z(1, 2);

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    t1(:, 6) = t1(:, 6) + 15;
    % info of not on board passengers
    f(:, 6) = f(:, 6) + 15;

    % info of all not arrived passengers
    t = [t1; f];
else % all the passengers arrived
    t = f;
end

else % bus has traveled but cannot pick up any passengers
    L = c(1:m, 1:3);

    % update bus location
    if c(m, 5) == 0
        z = c(m, 2:3);
    elseif abs(c(x, 2) - c(m, 2)) >= c(m, 5) && (c(x, 2) - c(m, 2)) >= 0
        z = [c(m, 2) + c(m, 5), c(m, 3)];
    elseif abs(c(x, 2) - c(m, 2)) >= c(m, 5) && (c(x, 2) - c(m, 2)) < 0
        z = [c(m, 2) - c(m, 5), c(m, 3)];
    elseif (c(x, 3) - c(m, 3)) >= 0
        z = [c(x, 2), c(m, 3) + c(m, 5) - abs(c(x, 2) - c(m, 2))];
    else
        z = [c(x, 2), c(m, 3) - abs(c(x, 2) - c(m, 2))];
    end

    % info of not arrived but has traveled passengers
    t1 = [];
end

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% info of not on board passengers
f(:, 6) = f(:, 6) + 15;

% no arrived passengers
g = [];

% info of all not arrived passengers
t = [t1; f];
end
else
    z = c(row, 2:3);
    L = c(1: row, 1:3);
    t = [];
    g = f;
    g(:, 2:3) = g(:, 4:5);
    [ig, ] = size(g);
    for i = 1: ig
        idg = find(c(:, 1) == g(i, 1), 1, 'last');
        g(i, 6) = g(i, 6) + 0.5 * c(idg, 6);
    end
end
end

```